Weighted Sampling without Replacement for Deep Top-k Classification

Dieqiao Feng [1] **Yuanqi Du** [1] **Carla P. Gomes** [1] **Bart Selman** [1]

Abstract

The top-k classification accuracy is a crucial metric in machine learning and is often used to evaluate the performance of deep neural networks. These networks are typically trained using the cross-entropy loss, which optimizes for top-1 classification and is considered optimal in the case of infinite data. However, in real-world scenarios, data is often noisy and limited, leading to the need for more robust losses. In this paper, we propose using the Weighted Sampling Without Replacement (WSWR) method as a learning objective for top-k loss. While traditional methods for evaluating **WSWR-based top-k loss** are computationally impractical, we show a novel connection between WSWR and Reinforcement Learning (RL) and apply well-established RL algorithms to estimate gradients. We compared our method with recently proposed top-k losses in various regimes of noise and data size for the prevalent use case of $k = 5$. Our experimental results reveal that our method consistently outperforms all other methods on the top-k metric for noisy datasets, has more robustness on extreme testing scenarios, and achieves competitive results on training with limited data.

1. Introduction

Classification, as a fundamental discipline within the field of machine learning, has undergone significant evolution in recent years, particularly with the emergence of problems involving hundreds or even thousands of classes. However, despite its importance, classification tasks present inherent challenges such as label confusion. This confusion can originate from a variety of factors, such as incorrect labeling and ambiguities that obfuscate the ground truth label even to a human expert. In the field of Reinforcement Learning (RL), this confusion can manifest in the initial stages of learning due to the poor policy of the RL agent (Silver et al., 2016; Feng et al., 2020). As a result, the metric of top-k classification accuracy has gained increasing importance, where the model must correctly identify the label from one of the top-k predictions. Conventionally, models are trained to optimize for top-1 accuracy, with top-5 and other metrics utilized solely for evaluation purposes. However, recent research has challenged this approach, proposing alternative methods such as the smoothed top-5 margin loss and differentiable sorting and ranking, which have been shown to demonstrate superior robustness in the presence of label noise and limited data when compared to the traditional top-1 cross-entropy loss (Lapin et al., 2016; Berrada et al., 2018; Petersen et al., 2022).

The design of a top-k loss function presents a significant challenge due to the lack of smoothness and sparsity of the derivatives utilized in backpropagation (Berrada et al., 2018). In this study, we propose a novel variant of top-k learning, which is inspired by the Weighted Sampling Without Replacement (WSWR) method (Hoeffding, 1963). Specifically, given a predicted positive weight vector $\mathbf{p} = (p_1, ..., p_n)$, where n represents the number of classes, we sample k items according to the weight vector $\mathbf{p}$ without replacement. It is important to note that after several items have been sampled, the sum of the remaining weights may not equal one, thus necessitating the renormalization of all weights into a probability distribution prior to each sampling. Given any true label y, we can derive the closed-form expression for the probability $P^{\mathbf{p}}_{k,y}$ of selecting y among the first k samples with weight vector $\mathbf{p}$. As such, it is natural to specify the learning objective as minimizing the negative log-likelihood of $P^{\mathbf{p}}_{k,y}$.

The evaluation of $P^{\mathbf{p}}_{k,y}$ and its gradient poses a significant challenge, as the closed-form expression of $P^{\mathbf{p}}_{k,y}$ consists of $\mathcal{O}(\binom{n-1}{k-1} \cdot k)$ terms, and there is currently no known polynomial algorithm for solving this problem (Ben-Hamou et al., 2018). In this work, we present a novel connection between the WSWR procedure and Reinforcement Learning (RL). Specifically, we designed a deterministic reward environment and established a link between the weight vector $\mathbf{p}$ and the stochastic policy of RL, such that the expected total

[*] Equal contribution [1] Department of Computer Science, Cornell University, Ithaca, U.S.. Correspondence to: Dieqiao Feng <dqfeng@cs.cornell.edu>, Yuanqi Du <yd392@cornell.edu>, Carla P. Gomes <gomes@cs.cornell.edu>, Bart Selman <selman@cs.cornell.edu>.

Proceedings of the 40th International Conference on Machine Learning, Honolulu, Hawaii, USA. PMLR 202, 2023. Copyright 2023 by the author(s).

rewards of the RL environment given the policy are identical to the closed-form expression of $P_{k,y}^{\mathbf{P}}$. As a result, various well-established RL optimization methods can be applied to estimate the gradient of $P_{k,y}^{\mathbf{P}}$.

The deterministic reward environment, however, presents a significant limitation in the form of sparse gradients, particularly when p_y is orders of magnitude smaller than other weights. This issue is particularly pronounced during the initial training stages, making the optimization process difficult. To address this limitation, we propose an improved learnable reward environment for WSWR, as well as various techniques for optimizing RL problems when reward functions are predicted by deep neural networks rather than fixed functions. These techniques are aimed at addressing the problem of sparse gradients, thereby enhancing the stability and efficiency of the training process.

We conducted an empirical evaluation of our method on the CIFAR-100 dataset with label noise, as well as subsets of the ImageNet-1K when training from scratch (Krizhevsky et al., 2009; Deng et al., 2009). We compared our method with multiple recent models, such as Bitonic Differentiable Sorting Network, NeuralSort, SoftSort, and SmoothTopKLoss (Berrada et al., 2018; Petersen et al., 2022; Grover et al., 2019; Prillo & Eisenschlos, 2020). In the noisy CIFAR-100 experiment, for top-5 accuracy, we demonstrate that our proposed method consistently outperforms **all** baselines, as well as the best two differentiable sorting and ranking methods when augmented by relaxing the assumption of a fixed k. On subsets of the ImageNet-1K dataset, we show that our method can achieve better accuracy, both for top-1 and top-5, as the ratio of training data decreases below 10%, while the performance improvements for subsets larger than 10% are limited. We also show our method consistently performs well on boundary and extreme test scenarios, while other top-k baselines experience significant performance drops and instability.

We summarize our contributions as follows:

- We derived a novel top-k loss based on weighted sampling without replacement.
- We proposed a novel problem remodeling technique to simplify the estimation of the gradient of the WSWR-based top-k loss.
- We empirically verified that it outperforms all other baseline methods on the top-5 metric for noisy datasets, has more robustness on extreme testing scenarios, and achieves competitive results when training with limited data.

2. Related Work

In the following section, an overview of recent state-of-the-art top-k methods is presented, which are subsequently evaluated in the experiment section.

In their study, Lapin et al. (2017) performed an in-depth analysis of single-label multi-class methods and presented a comprehensive study of efficient optimization algorithms for them. The authors demonstrated that cross-entropy is top-k calibrated for any value of k, which is a necessary condition for the classifier to be consistent with regard to the theoretically optimal top-k risk. In other words, cross-entropy satisfies an essential property for the optimal top-k classification decision for any k in the limit of infinite data. This finding may explain the good performance of cross-entropy on top-5 error on large-scale data sets. Additionally, the authors proposed a number of top-k loss functions and showed the possibility of further improvement for a specific value of k. Despite the thoroughness of the study, it should be noted that the experiments were conducted on linear models or pre-trained deep networks that were fine-tuned.

Berrada et al. (2018) posited that top-k losses are challenging to optimize as a result of their non-smooth nature and sparse derivatives. To mitigate these issues, the authors introduced additional smoothness into the top-k SVM loss and employed a polynomial algebra and divide-and-conquer approach to simplify the calculation. Through experimental evaluation, the authors showed that the smoothed top-k SVM loss outperforms cross-entropy in terms of both top-1 and top-5 accuracy on the CIFAR-100 dataset with label noise. Furthermore, for subsets of the ImageNet-1K dataset, the smoothed top-k SVM loss was found to be superior to cross-entropy in terms of top-5 accuracy, but only on 5% of the training data for top-1 accuracy.

Yang & Koyejo (2020) provided a theoretical analysis on the consistency of top-k surrogate losses, which relates to the convergence of the learned model to the population optimal prediction in the finite limit. They proposed a weighted top-k surrogate loss based on Bergman divergences. They conducted experiments on synthetic data to empirically support the theoretical analysis.

Petersen et al. (2022) proposed a relaxation method which instead of fixing k to be a constant value, draws k from a distribution P_k which may or may not depend on the confidence of specific data points or on the class label. Examples of distributions P_k are $[0.5, 0.0, 0.0, 0.0, 0.5]$ (half top-1 and half top-5) and $[0.2, 0.2, 0.2, 0.2, 0.2]$ (average over top-k for k ranging from 1 to 5). The authors found that relaxing k not only led to better top-5 accuracy, but also to improvements in top-1 accuracy. They evaluated their method on the smoothed top-k SVM loss, as well as NeuralSort (Grover et al., 2019), SoftSort (Prillo & Eisenschlos, 2020), and Differentiable Sorting Network (Petersen et al., 2021). NeuralSort relaxes permutation matrices to unimodal row-stochastic matrices by using the softmax of pairwise differences of cumulative sums of the top elements. SoftSort is a faster alternative to NeuralSort by simplifying the formulation and performs approximately equivalent to NeuralSort in the experiments. Differentiable Sorting Network is a continuous relaxation of the sorting network, which uses softmin and softmax instead of min and max when perturbing the values on the wires in each layer of the sorting network.

In addition to the aforementioned baselines, Fan et al. (2017) introduced a novel aggregate loss function that calculates the average of the k largest individual losses within a training dataset for the purpose of binary classification and regression. Lapin et al. (2015) extended the widely utilized multiclass support vector machine (SVM) to optimize for top-k error (Cortes & Vapnik, 1995). The proposed method employs a fast optimization technique based on an efficient projection onto the top-k simplex and demonstrated consistent improvement on the top-k metric across five datasets.

3. WSWR-based Top-k Loss

3.1. Weighted Sampling without Replacement

Fagin and Price proposed an experimental methodology referred to as Weighted Sampling Without Replacement (WSWR) (1978). This experiment involves drawing a random sample of size k from a population of n weighted items, where $1 \leq k \leq n$. The initial probability of drawing each item i is represented by p_i for $i = 1, ..., n$, and it is assumed that $\sum_i p_i = 1$. The process begins by selecting the first item, i_1, from the population. The probabilities of the remaining $n-1$ items are then renormalized such that they sum to 1, resulting in the probability of drawing item j becoming $p_j/(1-p_{i_1})$ for $j \neq i_1$. The process is repeated, selecting the next item, i_2, and renormalizing the probabilities of the remaining items, until k items have been sampled. Fagin and Price utilized this experimental methodology in their Monte Carlo evaluation of a combinatorial sum (Wong & Easton, 1980).

Let $\boldsymbol{\mathcal{I}} = (I_1, ..., I_k)$ denote the k-tuple of random variables of sampled indices, where $I_1, I_2, ..., I_k$ are sampled in order. The probability of each specific k-tuple $(i_1, ..., i_k)$ of distinct indices in $\{1, ..., n\}$ can be represented as follows:

$$\mathbb{P}((I_1, ..., I_k) = (i_1, ..., i_k)) = \prod_{j=1}^{k} \frac{p_{i_j}}{1 - \sum_{t=1}^{j-1} p_{i_t}}. \quad (1)$$

3.2. Top-k Classification

Given a data distribution $\mathcal{D}$, a classification model is trained to predict a positive weight vector $\mathbf{p} = (p_1, ..., p_n)$ (also known as a probability distribution) among n classes, for each pair of inputs (X, y) drawn from $\mathcal{D}$, where X is the input instance and $y \in \{1, ..., n\}$ is its corresponding label. The objective of WSWR-based top-k learning is to maximize the probability of sampling the true label y among the first k samples, based on the predicted weight vector $\mathbf{p}$. Specifically, the loss function can be defined as negative log-likelihood of the sampling probability:

$$\mathcal{L}(\mathbf{p}, y) = -\log \left(\sum_{y \in (i_1, ..., i_k)} \prod_{j=1}^{k} \frac{p_{i_j}}{1 - \sum_{t=1}^{j-1} p_{i_t}} \right). \quad (2)$$

The summation is taken over all k-tuples $(i_1, ..., i_k)$ of distinct indices that comprise true label y.

3.3. Computational Challenges

Upon initial analysis, the computation of $\mathcal{L}(\mathbf{p}, y)$ may seem computationally expensive. This is due to the presence of a summation over all possible k-tuples $(i_1, ..., i_k)$ that contain y, which have a cardinality of $\binom{n-1}{k-1} \cdot k$. In the case of ImageNet-1K, which comprises $1,000$ classes, the computation of the WSWR-based top-5 loss necessitates the evaluation of $\binom{999}{4} \cdot 5 \approx 2 \cdot 10^{11}$ terms for each individual training instance. This renders the approach practically infeasible. To address this computational challenge, we discovered a hidden connection between the problem and traditional reinforcement learning, and propose the application of well-established policy gradient algorithms as a means of providing a cheap way to approximate the real sampling probability and gradients.

4. Reinforcement Learning Reframing

4.1. Background

Reinforcement Learning (RL) is a general-purpose framework for decision-making (Kaelbling et al., 1996; Weng, 2018). Starting from some initial states S_0, the agent and environment interact through a sequence of actions and observed rewards over time t, where $t = 1, ..., T$. During this process, the agent accumulates knowledge about the environment, improves the current policy, and makes decisions on which action to take next in order to efficiently learn the optimal policy. The state, action, and reward at time step t are denoted as S_t, A_t, and R_t (may occasionally referred as s_t, a_t, r_t as well), respectively. The interaction sequence is fully described by one episode, also known as a trajectory, and the sequence ends at some terminal state S_T:

$$S_0, A_1, R_1, S_1, ..., S_T.$$

The model in RL is a descriptor of the environment. It enables the agent to learn or infer how the environment will interact with and provide new observations and intermediate rewards to the agent. The model consists of two main components: the transition probability function P and the reward function R. The objective of RL is to construct an artificial agent within the environment model to maximize expected future rewards.

In a single transition step, from any state s, the agent takes action a, leading to the next state s' and obtaining reward r. This is represented by the tuple (s, a, s', r). The transition function P records the probability of transitioning from state s to s' after taking action a and obtaining reward r:

$$P(s', r \mid s, a) = \mathbb{P}[S_t = s', R_t = r \mid S_{t-1} = s, A_t = a].$$

The policy of the agent is typically represented as a parameterized function $\pi_\theta(a \mid s)$. The objective is to learn a policy that maximizes the expected total rewards. Policy gradient methods aim to directly model and optimize the policy. The learning objective can be formalized as the maximization of expected total rewards:

$$H(\theta) = \mathbb{E}_{\pi_\theta}\left(\sum_{t=1}^{\infty} R_t \cdot \gamma^{t-1}\right),$$

where γ is a discount factor that reduces the importance of future rewards.

Policy Gradient Theorem. The Policy Gradient Theorem is a key concept in reinforcement learning that enables the computation of the gradient of the expected total rewards with respect to the policy's parameters, $\nabla_\theta H(\theta)$ (Sutton et al., 1999). Due to the trajectory sampling dependency on π_θ, directly computing the gradient is challenging. The Policy Gradient Theorem reformulates the gradient calculation by moving the derivatives inside the expectation:

$$\nabla_\theta H(\theta) \sim \mathbb{E}_{\pi_\theta}\left[Q^\pi(s, a) \nabla_\theta \ln \pi_\theta(a \mid s)\right],$$

where $Q(s, a)$ is the expected total rewards from the state s after taking the action a. Many policy gradient algorithms, such as REINFORCE and Advantage Actor-Critic (A2C), have been developed based on the Policy Gradient Theorem (Williams, 1992; Mnih et al., 2016). In this work, we will primarily focus on these two algorithms.

Throughout the remainder of this paper, we will only examine RL environments in which the transition function is deterministic given the previous state S_{t-1} and the selected action A_t, such that $S_t = P(S_{t-1}, A_t)$.

4.2. Deterministic Reward Environment for WSWR

This section presents a deterministic reward environment for the WSWR-based top-k sampling probability. The term "deterministic" comes from the fact that the reward function is fixed, as in traditional RL problems, as opposed to the learnable reward function as will be discussed in the subsequent Section 4.3.

As described in Equation 2, let's set $J(\theta)$ as the total sampling probability:

$$J(\theta) = \sum_{y \in (i_1, \ldots, i_k)} \prod_{j=1}^{k} \frac{p_{i_j}}{1 - \sum_{t=1}^{j-1} p_{i_t}}. \tag{3}$$

The ability to approximate both $J(\theta)$ and $\nabla_\theta J(\theta)$, where θ represents the parameters of the network, enables the approximation of the gradients of the loss function through the following equation:

$$\nabla_\theta \mathcal{L}(\mathbf{p}, y) = -\frac{\nabla_\theta J(\theta)}{J(\theta)}. \tag{4}$$

To achieve this, $J(\theta)$ can be rewritten as:

$$\begin{aligned} J(\theta) &= \sum_{y \in (i_1, \ldots, i_k)} \prod_{j=1}^{k} \frac{p_{i_j}}{1 - \sum_{t=1}^{j-1} p_{i_t}} \\ &= \sum_{(i_1, \ldots, i_k)} \left[\mathbb{1}_{y \in (i_1, \ldots, i_k)} \cdot \prod_{j=1}^{k} \frac{p_{i_j}}{1 - \sum_{t=1}^{j-1} p_{i_t}}\right] \\ &= \sum_{(i_1, \ldots, i_k)} \left[\left(\sum_{j=1}^{k} \mathbb{1}_{i_j = y}\right) \cdot \prod_{j=1}^{k} \frac{p_{i_j}}{1 - \sum_{t=1}^{j-1} p_{i_t}}\right]. \end{aligned} \tag{5}$$

If we view the k-tuple $(i_1, \ldots, i_k)$ as the sequence of actions of one episode in RL, $\mathbb{1}_{i_j = y}$ as the reward for selecting action i_j, and $p_{i_j}/(1 - \sum_{t=1}^{j-1} p_{i_t})$ as the probability of the current RL policy π to pick action i_j, then $\sum_{j=1}^{k} \mathbb{1}_{i_j = y}$ can be viewed as the total reward for one episode and $J(\theta)$ is actually calculating the expected total reward under the policy π. With this, we can use the Policy Gradient Theorem to estimate the gradient $\nabla_\theta J(\theta)$.

We formally define the state at time t, S_t to represent the set of remaining available indices and the actions to correspond to the selection of an item from the current state set. As a result, $A_t = S_{t-1}$, $S_0 = \{1, \ldots, n\}$, and the cardinality of S_t is $n - t$. At state S_{t-1}, for each action $a \in A_t$, the policy can be defined as $\pi(a \mid S_{t-1}) = p_a / \sum_{i \in S_{t-1}} p_i$, the state transition function is defined as $S_t = S_{t-1} \setminus \{a\}$, and the intermediate reward R_t is defined as $\mathbb{1}_{a=y}$. There is no discount factor applied to future rewards and the length of a single episode is fixed at k.

It can be easily verified $J(\theta)$, which is equivalent to the WSWR-based top-k sampling probability, is equal to the expected total reward of the above-defined RL environment:

$$J(\theta) = \mathbb{E}_{a \sim \pi}\left[\sum_{j=1}^{k} R_j\right]. \tag{6}$$

Consequently, maximizing the sampling probability is equivalent to maximizing the expected total rewards in a traditional RL setting. This equivalence enables the use of well-established on-policy policy gradient algorithms, such as REINFORCE and A2C, to approximate the true gradients and optimize the predicted sampling weights $\mathbf{p} = (p_1, ..., p_n)$.

$J(\theta)$ can be estimated by sampling multiple trajectories and calculating the average total reward for each trajectory. To estimate the gradient, we generate a set of m trajectories, denoted as $\tau_u = (s_0, a_1, s_1, ..., s_k)$, for each pair of output $\mathbf{p}$ and true label y, where $1 \leq u \leq m$. These trajectories are generated according to the policy $\pi(a \mid s)$, and m is a hyperparameter that controls the tradeoff between computational cost and gradient quality. As the number of sampled trajectories increases, the computational cost also increases, resulting in more accurate gradients.

To estimate the gradient, REINFORCE algorithm is applied and the gradient is estimated as:

$$\begin{aligned}\nabla_\theta J(\theta) &= \mathbb{E}_\pi \left[\sum\nolimits_{j=t}^{k} R_j \nabla_\theta \ln \pi_\theta(A_t \mid S_{t-1})\right] \\ &\approx \frac{1}{m} \cdot \sum_{u=1}^{m} \sum_{t=1}^{k} \sum_{j=t}^{k} R_j \nabla_\theta \ln \pi_\theta(A_t \mid S_{t-1}) \\ &= \frac{1}{m} \cdot \sum_{u=1}^{m} \sum_{t=1}^{k} \sum_{j=t}^{k} R_j \nabla_\theta \ln \left(\frac{p_{a_t}}{\sum_{i \in S_{t-1}} p_i}\right). \end{aligned} \tag{7}$$

4.3. Learnable Reward Environment for WSWR

The deterministic reward environment for WSWR, as previously discussed, presents one major limitation in its optimization process. During the initial stages of training, the weight p_y for the true label may be orders of magnitude smaller than the other top $k-1$ weights, resulting in a near-zero probability of sampling the true label y. This, in turn, implies that the total reward will be zero for almost all trajectories and gradients will always be zero if the true label is not sampled, as described in Equation 7.

To address this limitation, we proposed a trick that removes the action of true label y from the action space and incorporates it into the reward function to encourage the agent to increase p_y through maximizing total rewards. This approach is referred to as the learnable reward environment for WSWR. Specifically, we can further rewrite $J(\theta)$ to

$$\begin{aligned} J(\theta) = p_y + \sum_{y \notin (i_1, ..., i_{k-1})} &\left[\sum_{u=1}^{k-1} \left(\frac{p_y}{1 - \sum_{t=1}^{u} p_{i_t}} \right.\right. \\ &\left. \cdot \prod_{j=1}^{u} \frac{1 - p_y - \sum_{t=1}^{j-1} p_{i_t}}{1 - \sum_{t=1}^{j-1} p_{i_t}} \right) \\ &\left. \cdot \prod_{j=1}^{k-1} \frac{p_{i_j}}{1 - p_y - \sum_{t=1}^{j-1} p_{i_t}} \right] \end{aligned} \tag{8}$$

The whole deduction of Equation 8 can be found in Appendix A.

The outermost summation is taken over all $(k-1)$-tuples of distinct indices that exclude the true label y. The length of a single episode is now fixed at $k-1$, as only actions that are not the true label are being sampled. The state representation $\hat{S}_t$ also includes the remaining indices, with the change that $\hat{S}_0 = \{1, ..., y-1, y+1, ..., n\}$. The action set $\hat{A}_t$ now comprises all indices $i \in \hat{S}_{t-1}$, resulting in $\hat{A}_t = \hat{S}_{t-1}$. For each action $a \in \hat{A}_t$, the policy for state $\hat{S}_{t-1}$ is modified as $\hat{\pi}(a \mid \hat{S}_{t-1}) = p_a / \sum_{i \in \hat{S}_{t-1}} p_i$ and the state transition function is defined as $\hat{S}_t = \hat{S}_{t-1} \setminus \{a\}$. The expression

$$\sum_{u=1}^{k-1} \left(\frac{p_y}{1 - \sum_{t=1}^{u} p_{i_t}} \cdot \prod_{j=1}^{u} \frac{1 - p_y - \sum_{t=1}^{j-1} p_{i_t}}{1 - \sum_{t=1}^{j-1} p_{i_t}} \right)$$

in Equation 8 can be interpreted as the total reward for one episode. Therefore, we can define $\hat{R}_t$ for each timestamp as follows:

$$\begin{aligned} \hat{R}_t &= \frac{p_y}{1 - \sum_{t=1}^{t} p_{i_t}} \cdot \prod_{j=1}^{t} \frac{1 - p_y - \sum_{t=1}^{j-1} p_{i_t}}{1 - \sum_{t=1}^{j-1} p_{i_t}} \\ &= \frac{p_y}{p_y + \sum_{i \in \hat{S}_t} p_i} \cdot \prod_{j=1}^{t} \left(1 - \frac{p_y}{p_y + \sum_{i \in \hat{S}_{t-1}} p_i}\right). \end{aligned} \tag{9}$$

The starting reward, $\hat{R}_0 = p_y$, is obtained without taking any action and is immediately added to $J(\theta)$. As a result, the gradient will be non-zero even if p_y is significantly smaller than the other sampling weights.

So the total sampling probability $J(\theta)$ again matches the expected total rewards in the learnable reward environment defined above, as represented by Equation 10:

$$J(\theta) = \hat{R}_0 + \mathbb{E}_{a \sim \hat{\pi}} \left[\sum\nolimits_{j=1}^{k-1} \hat{R}_j\right]. \tag{10}$$

The calculation of the gradient $\nabla_\theta J(\theta)$ poses a challenge due to the fact that the reward function $\hat{R}$ is also a predicted value. There are two learnable terms in $J(\theta)$: the policy $\hat{\pi}_\theta$ and the reward $\hat{R}_\theta$. To overcome this challenge, we use the chain rule and fix one term as a constant while computing the gradient of the other, as demonstrated below:

$$\nabla_\theta J(\theta) = \nabla_\theta \hat{\pi}_\theta + \nabla_\theta \hat{R}_\theta \tag{11}$$

$$\nabla_\theta \hat{\pi}_\theta = \mathbb{E}_{\hat{\pi}} \left[\sum_{j=t}^{k-1} \hat{R}_j \nabla_\theta \ln \hat{\pi}_\theta (\hat{A}_t \mid \hat{S}_{t-1}) \right] \tag{12}$$

$$\nabla_\theta \hat{R}_\theta = \nabla_\theta \left[\hat{R}_0^\theta + \mathbb{E}_{\hat{\pi}} \sum_{j=1}^{k-1} \hat{R}_j^\theta \right] \tag{13}$$

$$= \nabla_\theta \hat{R}_0^\theta + \mathbb{E}_{\hat{\pi}} \sum_{j=0}^{k-1} \nabla_\theta \hat{R}_j^\theta \tag{14}$$

Equation 12 is a result of the Policy Gradient Theorem, while Equation 13 is a consequence of the exchange between the expectation and the gradient.

Estimating $J(\theta)$ requires sampling multiple trajectories and taking the average of total rewards. To estimate $\nabla_\theta J(\theta)$, m trajectories are sampled, and the gradients $\nabla_\theta \hat{\pi}_\theta$ and $\nabla_\theta \hat{R}_\theta$ are estimated separately.In this paper, we will adopt the use of the same set of trajectories, as the sampling procedure is computationally expensive.

4.4. Baseline for REINFORCE

According to the theory of the A2C, finding a suitable baseline $V(\hat{S}_{t-1})$ that estimates the future return $\sum_{j=t}^{k-1} \hat{R}_j$ and subtracting the baseline from the future return can significantly reduce the variance of gradients, thereby increasing the training speed. In our problem setting, the dynamic nature of the RL environment requires the derivation of a closed-form expression for $V(\hat{S}_{t-1})$. Additionally, the computational efficiency of $V(\hat{S}_{t-1})$ is a crucial factor, as a complex formula can impede training speed. Taking these factors into account, we have formulated a computationally simple baseline approximator, which will be utilized in all the experiments:

$$\begin{aligned} V(\hat{S}_{t-1}) = & \left[1 - \left(\frac{\sum_{i \in \hat{S}_t} p_i}{p_y + \sum_{i \in \hat{S}_t} p_i} \right)^{k-t} \right] \\ & \cdot \prod_{j=1}^{t} \left(1 - \frac{p_y}{p_y + \sum_{i \in \hat{S}_{t-1}} p_i}\right). \end{aligned} \tag{15}$$

5. Experiments

5.1. Baseline Setup

We evaluate our proposed WSWR-based top-k loss against five baseline methods: Cross Entropy (CE), Smoothed Top-k SVM (SSVM), Relaxed-k Bitonic Differentiable Sorting Network (RBitonic), Relaxed-k NeuralSort (RNeuralSort), and Relaxed-k SoftSort (RSoftSort) (Berrada et al., 2018; Petersen et al., 2022). The default value for k is set to 5, and the distribution $P_k = [0.2, 0.2, 0.2, 0.2, 0.2]$ is used for the relaxed-k losses. The hyperparameters for each loss are set as per Berrada et al. (2018); Petersen et al. (2022). By default, the number of sampled trajectories for our WSWR-based top-k loss is 100, which may vary with further notice.

5.2. CIFAR-100 with Label Noise

Dataset. The CIFAR-100 dataset comprises $50,000$ training images and $10,000$ test images. All images within the dataset are colored and possess a resolution of 32x32 pixels. The dataset is comprised of 100 distinct classes, each containing 600 images, with 500 images designated for training and 100 images designated for testing. These 100 classes are further grouped into 20 superclasses, referred to as coarse labels, and each coarse label is comprised of five fine labels. For example, the coarse label "flowers" is composed of the fine label "orchids," "poppies," "roses," "sunflowers," and "tulips." To enhance the diversity of the training dataset, a standard technique of data augmentation is employed, which includes random cropping with padding of 4 pixels, random horizontal flipping, and random rotation of up to 15 degrees (Shorten & Khoshgoftaar, 2019; Perez & Wang, 2017). Subsequently, all images are normalized channel-wise before being fed into the network.

In order to introduce additional noise into the training dataset, a randomization procedure is implemented, in which the label of each image is replaced by a random label within the same coarse class, with probability p. It should be noted that the randomized label may be identical to the original true label. This randomization process is performed once on the training dataset and the labels of each image are preserved throughout the entire training process. Therefore, with a probability of $p = 0$, the training dataset remains unmodified and with $p = 1$, each image will have a chance of 80% of having a wrong label. Through this method, a perfect top-5 classifier can still attain 100% accuracy on the top-5 metric by consistently predicting all five fine labels within the same coarse class as the true label.

Training details. To evaluate our methods, we used the architecture ResNet-18 from He et al. (2016) and train it from scratch. We used stochastic gradient descent with momentum 0.9 and weight decay 0.001 as the optimizer. We trained the networks for 200 total epochs with batch size 128. The initial learning rate was set to 0.1 and decayed by 0.2 after epoch 60, 120, and 160. In addition, we also added 1 warm-up epoch at the beginning of training to stabilize the initial training. All experiments were performed using Nvidia V100 GPUs.

Results. Table 1 demonstrates the test accuracy of CIFAR-

Table 1. The test accuracy of CIFAR-100 was evaluated under varying levels of label noise, with results averaged over three distinct seeds. The top-5 metric demonstrates that WSWR, our proposed method, consistently outperformed **all** other baseline methods across **all** noise levels. Furthermore, it is noteworthy that WSWR was the **only** loss function that exhibited superior performance in comparison to the traditional cross-entropy loss when the level of noise was small. In terms of the top-1 metric, WSWR exhibited the highest prediction accuracy within the noise range of $[0.2, 0.4]$. (Best performances are **bolded** and second best are underlined.)

Noise	Top-1 Accuracy (%)						Top-5 Accuracy (%)					
Level	CE	SVM	RBitonic	RNeuralSort	RSoftSort	WSWR (ours)	CE	SVM	RBitonic	RNeuralSort	RSoftSort	WSWR (ours)
0.0	**76.470**	68.177	68.790	71.720	69.850	71.633	93.417	92.890	81.117	91.983	89.863	**94.043**
0.1	**71.530**	63.463	55.703	67.157	68.427	70.497	90.847	90.767	79.030	90.023	86.610	**92.513**
0.2	66.187	58.043	42.683	62.223	67.137	**68.803**	87.983	88.793	64.990	88.733	85.103	**91.080**
0.3	61.343	52.293	52.833	58.583	65.683	**66.623**	86.310	86.970	78.977	87.870	84.763	**89.803**
0.4	55.380	44.747	54.270	53.727	63.490	**63.520**	84.180	86.230	76.300	87.140	84.457	**89.247**
0.5	48.713	35.747	45.893	47.133	**60.600**	60.020	81.207	85.410	73.637	86.103	83.973	**88.047**
0.6	42.483	25.677	45.707	43.027	**55.577**	55.467	79.737	84.943	75.817	85.750	83.857	**87.763**
0.7	35.613	19.070	29.153	38.307	**48.627**	47.897	76.533	84.240	60.503	85.017	83.513	**86.867**
0.8	29.447	17.817	32.423	35.640	**40.250**	37.720	74.103	84.063	69.877	84.773	83.603	**85.893**
0.9	22.660	17.307	22.787	**29.283**	26.373	23.343	70.683	83.833	68.643	84.290	83.547	**85.453**
1.0	16.073	17.267	15.543	**17.563**	17.210	17.377	65.283	83.983	67.107	83.933	83.450	**84.760**
Speed	29.300	19.102	17.401	27.564	29.029	19.713	29.300	19.102	17.401	27.564	29.029	19.713

Table 2. Speed and scaling test for multiple choices of the number of sampled trajectories m for WSWR on CIFAR-100 with the noise level of 0.5. The results show that increasing m consistently improved both top-1 and top-5 accuracy for WSWR, but at the cost of a significant reduction in training speed. As shown in the table, WSMR-$m10$ has comparable top-1 and top-5 accuracies to the best loss, while still maintaining a competitive training speed compared to the cross-entropy loss. (Best performances are **bolded** and second best are underlined.)

	Training Speed (batch per sec)	Top-1 (%)	Top-5 (%)
CE	29.300	48.713	81.207
SVM	19.102	35.747	85.410
RBitonic	17.401	45.893	73.637
RNeuralSort	27.564	47.133	86.103
RSoftSort	29.029	60.600	83.973
WSWR-$m1$	26.320	56.607	87.733
WSWR-$m10$	24.975	59.757	88.027
WSWR-$m100$	19.713	60.020	88.047
WSWR-$m1000$	5.288	60.270	88.247
WSWR-$m10000$	0.744	**60.732**	**88.424**

Table 3. We tested the robustness and stability of the top-50 losses by setting k to 50 and noise level to 1.0. Our results indicate that all other top-k losses exhibit strong instability, including a significant decrease in top-1 or top-50 accuracy. In contrast, WSWR (our method) performed consistently and achieved the highest top-50 accuracy. Additionally, our top-1 accuracy was also the most competitive when compared to cross-entropy. (Best performances are **bolded** and second best are underlined.)

	Training Speed (batch per sec)	Top-1 (%)	Top-50 (%)
CE	29.345	**15.980**	96.155
SVM	4.573	2.750	85.210
RBitonic	14.440	2.565	64.425
RNeuralSort	24.080	0.025	98.950
RSoftSort	26.284	4.790	98.470
WSWR	10.345	11.237	**99.203**

100 under varying levels of label noise. Our proposed method, WSWR, consistently outperforms **all** other baseline methods in **all** levels of label noise when evaluating the top-5 metric. It is noteworthy that WSWR is the **only** loss that achieves higher top-5 accuracy than the cross-entropy loss when the noise level is less than 0.1.

In terms of the top-1 metric, we found that when the noise level is less than 0.5, WSWR outperforms all other top-k losses and even the cross-entropy loss when the noise level is larger than 0.1. As the noise level increases above 0.5, sort-based losses achieve slightly higher top-1 accuracy, though the margin of difference is relatively small.

Speed and scaling test. In addition to accuracy, the speed of loss functions is an important metric to consider, as top-k losses often significantly increase the computational cost compared to the cross-entropy loss. Furthermore, we aimed to investigate how the accuracy scales with the number of sampled trajectories m.

To this end, we conducted a speed and scaling test for multiple choices of m for WSWR with a fixed label noise level of 0.5. The results, presented in Table 2, demonstrate that the incremental increase of m can consistently improve both top-1 and top-5 accuracy, however with a trade-off of reduction in speed. As m increases, WSWR reaches the highest accuracies for both metrics. The accuracy gain after increasing m above 100 is marginal and less noticeable. Specifically, WSWR-$m10$ achieves comparable top-1 and top-5 accuracies to the best loss, while still maintaining a competitive training speed compared to the cross-entropy loss.

Boundary condition test. The top-k loss calculation typically involves a large number of terms, specifically $\mathcal{O}(\binom{n}{k})$, where n is the number of classes. Various simplifying techniques are employed to reduce computational demands. In practice, a common value for k is 5, as it is relatively small compared to n and does not significantly affect floating precision or training speed. In this experiment, k was set to 50, where $\binom{n}{k}$ reaches its maximum, to test the performance of top-k losses in extreme cases in terms of training speed, numerical stability, and prediction accuracy. The noise level was set to 1.0 to maximize the level of noise in the training dataset.

As seen in Table 3, other top-k losses exhibit strong instability in this extreme test case, while WSWR maintains consistent performances. The significant performance drop of other top-k losses may be due to improper hyperparameter settings, as these losses have 3 to 5 hyperparameters. This highlights the advantage of WSWR, as it only has one hyperparameter (the number of sampled trajectories), which is more generalizable across various testing scenarios.

5.3. Subsets of ImageNet-1K Training

The ImageNet-1K dataset is comprised of over 1.28 million training images and 50,000 validation images, organized into 1000 distinct categories, including objects, scenes, and animals. Each category contains a varying number of images, ranging from several hundred to thousands. The images were sourced from the Internet and annotated by human labelers, resulting in a dataset that presents a degree of ambiguity and noise within the labels. For experimentation, subsets of varying sizes are extracted from the 1.28 million training images, and the unmodified validation dataset of 50,000 images is used for testing. To augment the training set, standard data augmentation methods are employed, such as random resized crop to 224x224 and random horizontal flipping. All validation images are resized to 256x256 and undergo a center crop to a size of 224x224. Prior to being fed into the networks, all images undergo channel-wise normalization.

In order to construct subsets of the training dataset, experiments were conducted in the following ratios: 100% (1.28M images), 50% (640K images), 25% (320K images), 10% (128K images), and 5% (64K images). To maintain consistency with the original dataset, the proportion of each class within the subsets was kept constant. Additionally, to ensure that each run of the experiment would utilize the same subset of training images with the same ratio, the random seed was fixed for all experiments.

Training details. We used the architecture ResNet-50 from He et al. (2016) and trained all networks from scratch. We used stochastic gradient descent with momentum 0.9 and weight decay 0.0001 as the optimizer. We trained the networks for 90 total epochs with batch size 256. The initial learning rate was set to 0.1 and decayed by 0.1 after epoch 30 and 60. In addition, we also added 1 warm-up epoch at the beginning of training to stabilize the initial training. All experiments were performed using Nvidia A100 GPUs.

Table 4. The accuracies of ImageNet-1K were tested with different ratios of the training dataset. Cross-entropy has better top-1 accuracy than top-k losses, except at a ratio of 5%, where overfitting is a significant problem. Our method still consistently achieves the second best accuracy for the top-1 metric. For the top-5 metric, WSWR-$m100$ performs worse than SSVM, while WSWR-$m1000$ outperforms CE and SSVM. The study highlights the significance of using a higher number of sampled trajectories m for WSWR to achieve a more accurate gradient estimate, as Imagenet-1K has many more classes than CIFAR-100. (Best performances are **bolded** and second best are underlined.)

	Top-1 Accuracy (%)				Top-5 Accuracy (%)			
ratio of dataset	CE	SSVM	WSWR $m100$	WSWR $m1000$	CE	SSVM	WSWR $m100$	WSWR $m1000$
100%	**75.76**	68.50	69.53	69.59	**92.79**	92.10	92.29	92.33
50%	**71.59**	64.77	65.77	65.91	**90.16**	89.72	89.78	89.95
25%	**62.06**	59.54	60.14	60.32	85.92	86.02	88.12	**89.20**
10%	**51.03**	47.69	46.26	47.81	75.27	76.05	73.91	**76.87**
5%	34.39	35.48	34.08	**35.56**	58.56	63.92	60.62	**64.12**

Result. Table 4 presents the results of the ImageNet-1K experiment, where the validation accuracy was evaluated under varying ratios of the training dataset. The family of sort-based top-k losses was not included in this experiment, as Petersen et al. (2022) fine-tuned on state-of-the-art models for ImageNet-1K, rather than training from scratch. Unlike the marginal difference in accuracy observed between WSWR-$m100$ and WSWR-$m1000$ in the CIFAR-100 experiment, Table 4 demonstrates that a larger number of sampled trajectories play a crucial role in achieving higher top-1 and top-5 accuracies. One possible explanation for this is that as the number of classes increases, the length of trajectories, which is fixed at $k-1=4$, becomes relatively small in comparison to the total number of classes. As a result, too few sampled trajectories may not cover a sufficient number of indices, leading to gradients for most of the predicted weights being zero. The training speed reduction when increasing m for ImageNet-1K is less significant compared to CIFAR-100 since larger network architecture will lower the impact of the computation of loss functions. The performance gain of using WSWR over cross-entropy is more significant as the ratio of training data becomes smaller. This confirms the theoretical result that when the training data is sufficient, cross-entropy is the best choice.

6. Conclusion

We introduced a new Weighted Sampling Without Replacement (WSWR) based top-k loss. By uncovering a novel link between WSWR and reinforcement learning, we developed an efficient method for estimating the gradient. Through empirical testing, we found that the proposed WSWR-based top-k loss significantly improves performance across various datasets and test scenarios, especially for the top-5 metric on noisy datasets. Additionally, we evaluated the robustness of various losses in extreme test cases where k is close to the half of the total number of classes, and found that WSWR was the only top-k loss that performed consistently.

References

Ben-Hamou, A., Peres, Y., and Salez, J. Weighted sampling without replacement. *Brazilian Journal of Probability and Statistics*, 32(3):657–669, 2018.

Berrada, L., Zisserman, A., and Kumar, M. P. Smooth loss functions for deep top-k classification. *arXiv preprint arXiv:1802.07595*, 2018.

Cortes, C. and Vapnik, V. Support-vector networks. *Machine learning*, 20:273–297, 1995.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pp. 248–255. Ieee, 2009.

Fagin, R. and Price, T. G. Efficient calculation of expected miss ratios in the independent reference model. *SIAM Journal on Computing*, 7(3):288–297, 1978.

Fan, Y., Lyu, S., Ying, Y., and Hu, B. Learning with average top-k loss. *Advances in neural information processing systems*, 30, 2017.

Feng, D., Gomes, C. P., and Selman, B. A novel automated curriculum strategy to solve hard sokoban planning instances. *Advances in Neural Information Processing Systems*, 33:3141–3152, 2020.

Grover, A., Wang, E., Zweig, A., and Ermon, S. Stochastic optimization of sorting networks via continuous relaxations. *arXiv preprint arXiv:1903.08850*, 2019.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Hoeffding, W. Probability inequalities for sums of bounded random variables. *Journal of the American statistical association*, 58(301):13–30, 1963.

Kaelbling, L. P., Littman, M. L., and Moore, A. W. Reinforcement learning: A survey. *Journal of artificial intelligence research*, 4:237–285, 1996.

Krizhevsky, A., Hinton, G., et al. Learning multiple layers of features from tiny images. 2009.

Lapin, M., Hein, M., and Schiele, B. Top-k multiclass svm. *Advances in neural information processing systems*, 28, 2015.

Lapin, M., Hein, M., and Schiele, B. Loss functions for top-k error: Analysis and insights. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1468–1477, 2016.

Lapin, M., Hein, M., and Schiele, B. Analysis and optimization of loss functions for multiclass, top-k, and multilabel classification. *IEEE transactions on pattern analysis and machine intelligence*, 40(7):1533–1554, 2017.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., and Kavukcuoglu, K. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pp. 1928–1937. PMLR, 2016.

Perez, L. and Wang, J. The effectiveness of data augmentation in image classification using deep learning. *arXiv preprint arXiv:1712.04621*, 2017.

Petersen, F., Borgelt, C., Kuehne, H., and Deussen, O. Differentiable sorting networks for scalable sorting and ranking supervision. In *International Conference on Machine Learning*, pp. 8546–8555. PMLR, 2021.

Petersen, F., Kuehne, H., Borgelt, C., and Deussen, O. Differentiable top-k classification learning. In *International Conference on Machine Learning*, pp. 17656–17668. PMLR, 2022.

Prillo, S. and Eisenschlos, J. Softsort: A continuous relaxation for the argsort operator. In *International Conference on Machine Learning*, pp. 7793–7802. PMLR, 2020.

Shorten, C. and Khoshgoftaar, T. M. A survey on image data augmentation for deep learning. *Journal of big data*, 6(1):1–48, 2019.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

Sutton, R. S., McAllester, D., Singh, S., and Mansour, Y. Policy gradient methods for reinforcement learning with function approximation. *Advances in neural information processing systems*, 12, 1999.

Weng, L. A (long) peek into reinforcement learning. *lilianweng.github.io*, 2018. URL https://lilianweng.github.io/posts/2018-02-19-rl-overview/.

Williams, R. J. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Reinforcement learning*, pp. 5–32, 1992.

Wong, C.-K. and Easton, M. C. An efficient method for weighted sampling without replacement. *SIAM Journal on Computing*, 9(1):111–113, 1980.

Yang, F. and Koyejo, S. On the consistency of top-k surrogate losses. In *International Conference on Machine Learning*, pp. 10727–10735. PMLR, 2020.

A. Derivation of Equation 8

$$
\begin{aligned}
J(\theta) &= \sum_{y\in(i_1,\ldots,i_k)} \prod_{j=1}^{k} \frac{p_{i_j}}{1-\sum_{t=1}^{j-1} p_{i_t}} \\
&= \sum_{y\notin(i_1,\ldots,i_{k-1})} \sum_{u=0}^{k-1} \left[\left(\prod_{j=1}^{u} \frac{p_{i_j}}{1-\sum_{t=1}^{j-1} p_{i_t}} \right) \cdot \frac{p_y}{1-\sum_{t=1}^{u} p_{i_t}} \cdot \prod_{j=u+1}^{k-1} \frac{p_{i_j}}{1-p_y-\sum_{t=1}^{j-1} p_{i_t}} \right] \\
&= \sum_{y\notin(i_1,\ldots,i_{k-1})} \sum_{u=0}^{k-1} \left[\left(\prod_{j=1}^{u} \frac{p_{i_j}}{1-p_y-\sum_{t=1}^{j-1} p_{i_t}} \cdot \frac{1-p_y-\sum_{t=1}^{j-1} p_{i_t}}{1-\sum_{t=1}^{j-1} p_{i_t}} \right) \cdot \frac{p_y}{1-\sum_{t=1}^{u} p_{i_t}} \cdot \prod_{j=u+1}^{k-1} \frac{p_{i_j}}{1-p_y-\sum_{t=1}^{j-1} p_{i_t}} \right] \\
&= \sum_{y\notin(i_1,\ldots,i_{k-1})} \sum_{u=0}^{k-1} \left[\left(\prod_{j=1}^{u} \frac{1-p_y-\sum_{t=1}^{j-1} p_{i_t}}{1-\sum_{t=1}^{j-1} p_{i_t}} \right) \cdot \frac{p_y}{1-\sum_{t=1}^{u} p_{i_t}} \cdot \prod_{j=1}^{k-1} \frac{p_{i_j}}{1-p_y-\sum_{t=1}^{j-1} p_{i_t}} \right] \\
&= \sum_{y\notin(i_1,\ldots,i_{k-1})} \left[\sum_{u=0}^{k-1} \left(\frac{p_y}{1-\sum_{t=1}^{u} p_{i_t}} \cdot \prod_{j=1}^{u} \frac{1-p_y-\sum_{t=1}^{j-1} p_{i_t}}{1-\sum_{t=1}^{j-1} p_{i_t}} \right) \cdot \prod_{j=1}^{k-1} \frac{p_{i_j}}{1-p_y-\sum_{t=1}^{j-1} p_{i_t}} \right] \\
&= p_y + \sum_{y\notin(i_1,\ldots,i_{k-1})} \left[\sum_{u=1}^{k-1} \left(\frac{p_y}{1-\sum_{t=1}^{u} p_{i_t}} \cdot \prod_{j=1}^{u} \frac{1-p_y-\sum_{t=1}^{j-1} p_{i_t}}{1-\sum_{t=1}^{j-1} p_{i_t}} \right) \cdot \prod_{j=1}^{k-1} \frac{p_{i_j}}{1-p_y-\sum_{t=1}^{j-1} p_{i_t}} \right]
\end{aligned}
\tag{16}
$$