SMOOTH LOSS FUNCTIONS FOR DEEP TOP-K CLASSIFICATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Human labeling of data constitutes a long and expensive process. As a consequence, many classification tasks entail incomplete annotation and incorrect labels, while being built on a restricted amount of data. In order to handle the ambiguity and the label noise, the performance of machine learning models is usually assessed with top-k error metrics rather than top-1. Theoretical results suggest that to minimize this error, various loss functions, including cross-entropy, are equally optimal choices of learning objectives in the limit of infinite data. However, the choice of loss function becomes crucial in the context of limited and noisy data. Besides, our empirical evidence suggests that the loss function must be smooth and non-sparse to work well with deep neural networks. Consequently, we introduce a family of smoothed loss functions that are suited to top-k optimization via deep learning. The widely used cross-entropy is a special case of our family. Evaluating our smooth loss functions is computationally challenging: a naïve algorithm would require $\mathcal{O}(\binom{C}{k})$ operations, where C is the number of classes. Thanks to a connection to polynomial algebra and a divide-and-conquer approach, we provide an algorithm with a time complexity of $\mathcal{O}(kC)$. Furthermore, we present a novel and error-bounded approximation to obtain fast and stable algorithms on GPUs with single floating point precision. We compare the performance of the cross-entropy loss and our margin-based losses in various regimes of noise and data size. Our investigation reveals that our loss provides on-par performance with cross-entropy for $k = 1$, and is more robust to noise and overfitting for $k = 5$.

1 INTRODUCTION

In machine learning many classification tasks present inherent label confusion. The confusion can originate from a variety of factors, such as incorrect labeling, incomplete annotation, or some fundamental ambiguity that obfuscate the ground truth label even to a human expert. For example, consider the images from the ImageNet data set (Russakovsky et al., 2015) in Figure 1, which illustrate the aforementioned factors. To mitigate these issues, one may require the model to predict the k most likely labels, where k is typically very small compared to the total number of labels. Then the prediction is considered incorrect if all of its k labels differ from the ground truth, and correct otherwise. This is commonly referred to as the top-k error. Learning such models is a longstanding task in machine learning, and many loss functions for top-k error have been suggested in the literature.

In the context of correctly labeled large data, deep neural networks trained with cross-entropy have shown exemplary capacity to accurately approximate the data distribution. An illustration of this phenomenon is the performance attained by deep convolutional neural networks on the ImageNet challenge. Specifically, state-of-the-art models trained with cross-entropy yield remarkable success on the top-5 error, although cross-entropy is not tailored for top-5 error minimization. This phenomenon can be explained by the fact that cross-entropy is top-k calibrated for any k (Lapin et al., 2016), an asymptotic property which is verified in practice in the large data setting. However, in cases where only a limited amount of data is available, learning large models with cross-entropy can be prone to over-fitting on incomplete or noisy labels.

To alleviate the deficiency of cross-entropy, we present a new family of top-k classification loss functions for deep neural networks. Taking inspiration from multi-class SVMs, our loss creates a margin between the correct top-k predictions and the incorrect ones. Our empirical results show

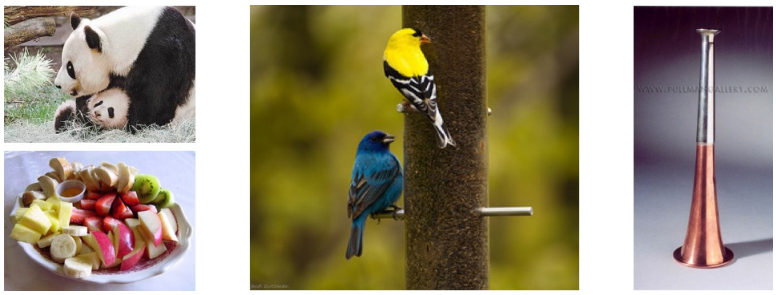


Figure 1: *Examples of images with label confusion, from the validation set of ImageNet. The top-left image is incorrectly labeled as "red panda", instead of "giant panda". The bottom-left image is labeled as "strawberry", although the categories "apple", "banana" and "pineapple" would be other valid labels. The center image is labeled as "indigo bunting", which is only valid for the lower bird of the image. The right-most image is labeled as a cocktail shaker, yet could arguably be a part of a music instrument (for example with label "cornet, horn, trumpet, trump"). Such examples motivate the need to predict more than a single label per image.*

that in their original form, such loss functions do not perform well in combination with deep neural networks. We believe that the reason for this is the lack of smoothness and the sparsity of the derivatives that are used in backpropagation. In order to overcome this difficulty, we smooth the loss with a temperature parameter. The evaluation of the smooth function and its gradient are challenging, as smoothing increases the naïve time complexity from $\mathcal{O}(C)$ to $\mathcal{O}(\binom{C}{k})$. With a connection to polynomial algebra and a divide-and-conquer method, we present an algorithm with $\mathcal{O}(kC)$ time complexity and training time comparable to cross-entropy in practice. We provide insights for numerical stability of the forward pass. To deal with instabilities of the backward pass, we derive a novel and inexpensive approximation for which we prove an error bound. Our investigation reveals that our top-k loss outperforms cross-entropy in the presence of noisy labels or in the absence of large amounts of data. We further confirm that the difference of performance reduces with large correctly labeled data, which is consistent with known theoretical results.

2 Related Work

Top-k Loss Functions. The majority of the work on top-k loss functions has been applied to shallow models: Lapin et al. (2016) suggest a convex surrogate on the top-k loss; Fan et al. (2017) select the k largest individual losses in order to be robust to data outliers; Chang et al. (2017) formulate a truncated re-weighted top-k loss as a difference-of-convex objective and optimize it with the Concave-Convex Procedure (Yuille & Rangarajan, 2002); and Yan et al. (2017) propose to use a combination of top-k classifiers and to fuse their outputs.

Closest to our work is the extensive review of top-k loss functions for computer vision by Lapin et al. (2017). The authors conduct a study of a number of top-k loss functions derived from cross-entropy and hinge losses. Interestingly, they prove that for any k, cross-entropy is top-k calibrated, 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 to perform the optimal top-k classification decision for any k in the limit of infinite data. This may explain why cross-entropy performs well on top-5 error on large scale data sets. While thorough, the experiments are conducted on linear models, or pre-trained deep networks that are fine-tuned. For a more complete analysis, we wish to design loss functions that allow for the training of deep neural networks from a random initialization.

Smoothing. Smoothing is a helpful technique in optimization. In work closely related to ours, Lee & Mangasarian (2001) show that smoothing a binary SVM with a temperature parameter improves the theoretical convergence speed of their algorithm. Schwing et al. (2012) use a temperature parameter to smooth latent variables for structured prediction. Lapin et al. (2017) apply Yoreau-Mosida regularization to smooth their top-k surrogate losses.

Smoothing has also been applied in the context of deep neural networks. In particular, Zheng et al. (2015) and Clevert et al. (2016) both suggest to modify the non-smooth ReLU activation to improve the training. Gulcehre et al. (2017) suggest to introduce "mollifyers" to smooth the objective function by gradually increasing the difficulty of the optimization problem. Chaudhari et al. (2017) add a local entropy term to the loss to promote solutions with high local entropy. These smoothing techniques are used to speed up the optimization or improve generalization. In this work, we show that smoothing is necessary for the neural network to perform well in combination with our loss function. We hope that this insight can also help the design of losses for tasks other than top-k error minimization.

3 TOP-K SVM

3.1 BACKGROUND: MULTI-CLASS SVM

In order to build an intuition about top-k losses, we start with the simple case $k = 1$, namely multi-class classification, where the output space is defined as $\mathcal{Y} = \{1, ..., C\}$. We suppose that a vector of scores per label $\mathbf{s} \in \mathbb{R}^C$, and a ground truth label $y \in \mathcal{Y}$ are both given. The vector $\mathbf{s}$ is the output of the model we wish to learn, for example a linear model or a deep neural network. The notation $\mathbb{1}$ will refer to the indicator function over boolean statements (1 if true, 0 if false).

Prediction. The prediction is given by any index with maximal score:

$$P(\mathbf{s}) \in \arg\max \mathbf{s}. \tag{1}$$

Loss. The classification loss incurs a binary penalty by comparing the prediction to the ground truth label. Plugging in equation (1), this can also be written in terms of scores $\mathbf{s}$ as follows:

$$\Lambda(\mathbf{s}, y) \triangleq \mathbb{1}(y \neq P(\mathbf{s})) = \mathbb{1}(\max_{j \in \mathcal{Y}} s_j > s_y). \tag{2}$$

Surrogate. The loss of equation (2) is not amenable to optimization, as it is not even continuous in $\mathbf{s}$. To overcome this difficulty, a typical approach in machine learning is to resort to a surrogate loss that provides a continuous upper bound on Λ. Crammer & Singer (2001) suggest the following upper bound on the loss, known as the multi-class SVM loss:

$$l(\mathbf{s}, y) = \max \left\{ \max_{j \in \mathcal{Y} \setminus \{y\}} \{s_j + 1\} - s_y, 0 \right\}. \tag{3}$$

In other words, the surrogate loss is zero if the ground truth score is higher than the largest other score by a margin of at least one. Otherwise it incurs a penalty which is linear in the difference.

Rescaling. Note that the value of 1 as a margin is an arbitrary choice, and can be changed to α for any $\alpha > 0$. This simply entails that we consider the cost Λ of a misclassification to be α instead of 1.

3.2 TOP-K CLASSIFICATION

We now generalize the above framework to top-k classification, where $k \in \{1, ..., C-1\}$. We use the following notation: for $p \in \{1, ..., C\}$, $s_{[p]}$ refers to the p-th largest element of $\mathbf{s}$, and $\mathbf{s}_{\setminus p}$ to the vector $(s_1, ..., s_{p-1}, s_{p+1}, ..., s_C) \in \mathbb{R}^{C-1}$ (that is, the vector $\mathbf{s}$ with the p-th element omitted). $\mathcal{Y}^{(k)}$ is the set of k-tuples with k distinct elements of $\mathcal{Y}$. Note that we use a bold font for a tuple $\bar{\mathbf{y}} \in \mathcal{Y}^{(k)}$ in order to distinguish it from a single label $\bar{y} \in \mathcal{Y}$.

Prediction. Given the scores $\mathbf{s} \in \mathbb{R}^C$, the top-$k$ prediction consists of any set of labels corresponding to the k largest scores:

$$P_k(\mathbf{s}) \in \left\{ \bar{\mathbf{y}} \in \mathcal{Y}^{(k)} : \forall\, i \in \{1, .., k\},\ s_{\bar{y}_i} \geq s_{[k]} \right\}. \tag{4}$$

Loss. The loss depends on whether y is part of the top-k prediction, which is equivalent to comparing the k-largest score with the ground truth score:

$$\Lambda_k(\mathbf{s}, y) \triangleq \mathbb{1}(y \notin P_k(\mathbf{s})) = \mathbb{1}(s_{[k]} > s_y). \tag{5}$$

Again, such a binary loss is not suitable for optimization. Thus we introduce a surrogate loss.

Surrogate. As pointed out in Lapin et al. (2015), there is a natural extension of the previous multi-class case:

$$l_k(\mathbf{s}, y) \triangleq \max\left\{\left(\mathbf{s}_{\setminus y} + \mathbf{1}\right)_{[k]} - s_y, 0\right\}. \tag{6}$$

This loss creates a margin between the ground truth and the k-th largest score, irrespectively of the values of the $(k-1)$-largest scores. Note that we retrieve the formulation of Crammer & Singer (2001) for $k = 1$.

Difficulty of the Optimization. The surrogate loss l_k of equation (6) suffers from two disadvantages that make it difficult to optimize: (i) it is not a smooth function of $\mathbf{s}$ – it is continuous but not differentiable – and (ii) its weak derivatives have at most two non-zero elements. Indeed at most two elements of $\mathbf{s}$ are retained by the $(\cdot)_{[k]}$ and $\max$ operators in equation (6). All others are discarded and thus get zero derivatives. When l_k is coupled with a deep neural network, the model typically yields poor performance, even on the training set. Similar difficulties to optimizing a piecewise linear loss have also been reported by Li et al. (2017) in the context of multi-label classification. We illustrate this in the next section.

We postulate that the difficulty of the optimization explains why there has been little work exploring the use of SVM losses in deep learning (even in the case $k = 1$), and that this work may help remedy it. We propose a smoothing that alleviates both issues (i) and (ii), and we present experimental evidence that the smooth surrogate loss offers better performance in practice.

3.3 Smooth Surrogate Loss

Reformulation. We introduce the following notation: given a label $\bar{y} \in \mathcal{Y}$, $\mathcal{Y}^{(k)}_{\bar{y}}$ is the subset of tuples from $\mathcal{Y}^{(k)}$ that include $\bar{y}$ as one of their elements. For $\bar{\mathbf{y}} \in \mathcal{Y}^{(k)}$ and $y \in \mathcal{Y}$, we further define $\Delta_k(\bar{\mathbf{y}}, y) \triangleq \mathbb{1}(y \notin \bar{\mathbf{y}})$. Then, by adding and subtracting the $k-1$ largest scores of $\mathbf{s}_{\setminus y}$ as well as s_y, we obtain:

$$\begin{aligned} l_k(\mathbf{s}, y) &= \max\left\{\left(\mathbf{s}_{\setminus y} + \mathbf{1}\right)_{[k]} - s_y, 0\right\}, \\ &= \max_{\bar{\mathbf{y}} \in \mathcal{Y}^{(k)}} \left\{\Delta_k(\bar{\mathbf{y}}, y) + \sum_{j \in \bar{\mathbf{y}}} s_j\right\} - \max_{\bar{\mathbf{y}} \in \mathcal{Y}^{(k)}_y} \left\{\sum_{j \in \bar{\mathbf{y}}} s_j\right\}. \end{aligned} \tag{7}$$

We give a more detailed proof of this in Appendix A.1. Since the margin can be rescaled without loss of generality, we rewrite l_k as:

$$l_k(\mathbf{s}, y) = \max_{\bar{\mathbf{y}} \in \mathcal{Y}^{(k)}} \left\{\Delta_k(\bar{\mathbf{y}}, y) + \frac{1}{k}\sum_{j \in \bar{\mathbf{y}}} s_j\right\} - \max_{\bar{\mathbf{y}} \in \mathcal{Y}^{(k)}_y} \left\{\frac{1}{k}\sum_{j \in \bar{\mathbf{y}}} s_j\right\}. \tag{8}$$

Smoothing. In the form of equation (8), the loss function can be smoothed with a temperature parameter $\tau > 0$:

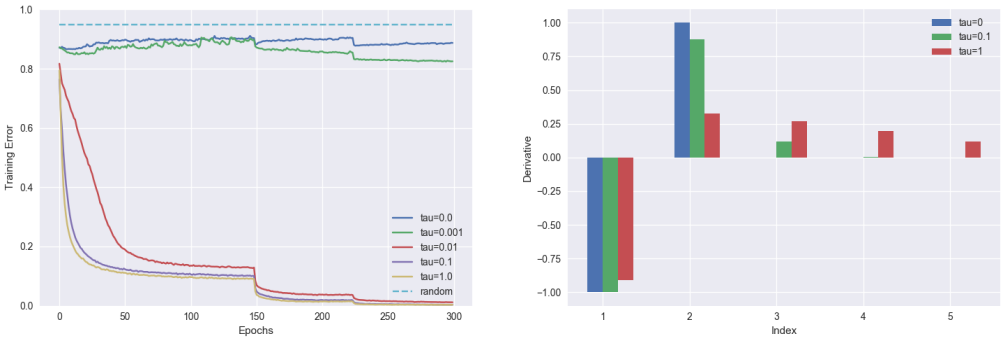
$$L_{k,\tau}(\mathbf{s}, y) = \tau \log\left[\sum_{\bar{\mathbf{y}} \in \mathcal{Y}^{(k)}} \exp\left(\frac{1}{\tau}\left(\Delta_k(\bar{\mathbf{y}}, y) + \frac{1}{k}\sum_{j \in \bar{\mathbf{y}}} s_j\right)\right)\right] - \tau \log\left[\sum_{\bar{\mathbf{y}} \in \mathcal{Y}^{(k)}_y} \exp\left(\frac{1}{k\tau}\sum_{j \in \bar{\mathbf{y}}} s_j\right)\right]. \tag{9}$$

Note that we have changed the notation to use $L_{k,\tau}$ to refer to the smooth loss. In what follows, we first outline the properties of $L_{k,\tau}$ and its relationship with cross-entropy. Then we show the empirical advantage of $L_{k,\tau}$ over its non-smooth counter-part l_k.

Properties of the Smooth Loss. The smooth loss $L_{k,\tau}$ has a few interesting properties. First, for any $\tau > 0$, $L_{k,\tau}$ is infinitely differentiable and has non-sparse gradients. Second, under mild conditions, when $\tau \to 0^+$, the non-maximal terms become negligible, therefore the summations collapse to maximizations and $L_{k,\tau} \to l_k$ in a pointwise sense (Proposition 2 in Appendix A.2). Third, $L_{k,\tau}$ is an upper bound on l_k if and only if $k = 1$ (Proposition 3 in Appendix A.3), but $L_{k,\tau}$ is, up to a scaling factor, an upper bound on Λ_k (Proposition 4 in Appendix A.4). This makes it a valid surrogate loss for the minimization of Λ_k.

Relationship with Cross-Entropy. We have previously seen that the margin can be rescaled by a factor of $\alpha > 0$. In particular, if we scale Δ by $\alpha \to 0^+$ and choose a temperature $\tau = 1$, it can be seen that $L_{1,1}$ becomes exactly the cross-entropy loss for classification. In that sense, $L_{k,\tau}$ is a generalization of the cross-entropy loss to: (i) different values of $k \geq 1$, (ii) different values of temperature and (iii) higher margins with the scaling α of Δ. For simplicity purposes, we will keep $\alpha = 1$ in this work.

Experimental Validation. In order to show how smoothing helps the training, we train a DenseNet 40-12 on CIFAR-100 from Huang et al. (2017) with the same hyper-parameters and learning rate schedule. The only difference with Huang et al. (2017) is that we replace the cross-entropy loss with $L_{5,\tau}$ for different values of τ. We plot the top-5 training error in Figure 2 (for each curve, the value of τ is held constant during training):



(a) *Top-5 training error for different values of τ on CIFAR-100. The dashed line $y = 0.95$ represents the base error for random predictions. The successive drops in the curves correspond to the decreases of the learning rate at epochs 150 and 225.*

(b) *Visualization of derivatives for $k = 1$ and various τ: each bar gives the value of the derivative (y-axis) for a given label (x-axis). These are the derivatives obtained with the synthetic data $\mathbf{s} = (0.5, 0.8, 0.3, 0, -0.5)$ and ground truth $y = 1$.*

Figure 2: *Influence of τ on the optimization. We confirm that smoothing helps the training of a neural network in Figure 2a, where a large enough value of τ greatly helps the performance on the training set. In Figure 2b, we visualize how the case $\tau = 0$ yields sparse derivatives, while increasing τ gives more information about all labels.*

We remark that the network exhibits good accuracy when τ is high enough (0.01 or larger). For τ too small, the model fails to converge to a good critical point. When τ is positive but small, the function is smooth but the gradients are numerically sparse (see Figure 2b), which suggests that the smoothness property is not sufficient and that non-sparsity is a key factor here.

4 Computational Challenges and Efficient Algorithms

4.1 Challenge

Experimental evidence suggests that it is beneficial to use $L_{k,\tau}$ rather than l_k to train a neural network. Moreover, at first glance, $L_{k,\tau}$ may appear prohibitively expensive to compute. Specifically, there are summations over $\mathcal{Y}^{(k)}$ and $\mathcal{Y}_y^{(k)}$, which have a cardinality of $\binom{C}{k}$ and $\binom{C}{k-1}$ respectively. For instance for ImageNet, we have $k = 5$ and $C = 1,000$, which amounts to $\binom{C}{k} \simeq 8.10^{12}$ terms to compute and sum over for each single sample, thereby making the approach practically infeasible. This is in stark contrast with l_k, for which the most expensive operation is to compute the k-th largest score of an array of size C, which can be done in $\mathcal{O}(C)$. To overcome this computational challenge, we will now reframe the problem and reveal its exploitable structure.

For a vector $\mathbf{e} \in \mathbb{R}^C$ and $i \in \{1, .., C\}$, we define $\sigma_i(\mathbf{e})$ as the sum of all products of i distinct elements of $\mathbf{e}$. Explicitly, $\sigma_i(\mathbf{e})$ can be written as $\sigma_i(\mathbf{e}) = \sum_{1 \leq j_1 < ... < j_i \leq C} e_{j_1} ... e_{j_i}$. The terms σ_i are known as the elementary symmetric polynomials. We further define $\sigma_0(\mathbf{e}) = 1$ for convenience.

We now re-write $L_{k,\tau}$ using the elementary symmetric polynomials, which appear naturally when separating the terms that contain the ground truth from the ones that do not:

$$\begin{aligned}
L_{k,\tau}(\mathbf{s}, y) &= \tau \log \Bigg[\sum_{\bar{\mathbf{y}} \in \mathcal{Y}^{(k)}} \exp\left(\Delta_k(\bar{\mathbf{y}}, y)/\tau\right) \prod_{j \in \bar{\mathbf{y}}} \exp(s_j/k\tau) \Bigg] \\
&\quad - \tau \log \Bigg[\sum_{\bar{\mathbf{y}} \in \mathcal{Y}_y^{(k)}} \prod_{j \in \bar{\mathbf{y}}} \exp(s_j/k\tau) \Bigg], \\
&= \tau \log \Bigg[\sum_{\bar{\mathbf{y}} \in \mathcal{Y}_y^{(k)}} \prod_{j \in \bar{\mathbf{y}}} \exp(s_j/k\tau) + \exp\left(1/\tau\right) \sum_{\bar{\mathbf{y}} \in \mathcal{Y}^{(k)} \setminus \mathcal{Y}_y^{(k)}} \prod_{j \in \bar{\mathbf{y}}} \exp(s_j/k\tau) \Bigg] \\
&\quad - \tau \log \Bigg[\sum_{\bar{\mathbf{y}} \in \mathcal{Y}_y^{(k)}} \prod_{j \in \bar{\mathbf{y}}} \exp(s_j/k\tau) \Bigg], \qquad (10) \\
&= \tau \log \Big[\exp(s_y/k\tau) \sigma_{k-1}(\exp(\mathbf{s}_{\setminus y}/k\tau)) + \exp\left(1/\tau\right) \sigma_k(\exp(\mathbf{s}_{\setminus y}/k\tau)) \Big] \\
&\quad - \tau \log \Big[\exp(s_y/k\tau) \sigma_{k-1}(\exp(\mathbf{s}_{\setminus y}/k\tau)) \Big].
\end{aligned}$$

Note that the application of $\exp$ to vectors is meant in an element-wise fashion. The last equality of equation (10) reveals that the challenge is to efficiently compute σ_{k-1} and σ_k, and their derivatives for the optimization.

While there are existing algorithms to evaluate the elementary symmetric polynomials, they have been designed for computations on CPU with double floating point precision. For the most recent work, see Jiang et al. (2016). To efficiently train deep neural networks with $L_{k,\tau}$, we need algorithms that are numerically stable with single floating-point precision and that exploit GPU parallelization. In the next sections, we design algorithms that meet these requirements.

4.2 Forward Computation

We consider the general problem of efficiently computing (σ_{k-1}, σ_k). Our goal is to compute $\sigma_k(\mathbf{e})$, where $\mathbf{e} \in \mathbb{R}^n$ and $k \ll n$. Since this algorithm will be applied to $\mathbf{e} = \exp(\mathbf{s}_{\setminus y}/k\tau)$ (see equation (10)), we can safely assume $e_i \neq 0$ for all $i \in [\![1, n]\!]$.

The main insight of our approach is the connection of $\sigma_i(\mathbf{e})$ to the polynomial:

$$P \triangleq (X + e_1)(X + e_2)...(X + e_n). \qquad (11)$$

Indeed, if we expand P to $\alpha_0 + \alpha_1 X + ... + \alpha_n X^n$, Vieta's formula gives the relationship:

$$\forall i \in [\![0, n]\!], \quad \alpha_i = \sigma_{n-i}(\mathbf{e}). \qquad (12)$$

Therefore, it suffices to compute the coefficients α_{n-k} to obtain the value of $\sigma_k(\mathbf{e})$. To compute the expansion of P, we can use a divide-and-conquer approach with polynomial multiplications when merging two branches of the recursion.

This method computes all $(\sigma_i)_{1 \leq i \leq n}$ instead of the only $(\sigma_i)_{k-1 \leq i \leq k}$ that we require. Since we do not need $\sigma_i(\mathbf{e})$ for $i > k$, we can avoid computations of all coefficients for a degree higher than $n - k$. For ImageNet, we have $k = 5$ and $n = 1,000$, therefore we have to compute coefficients up to a degree 995 instead of 1,000, which is a small improvement. To turn $k \ll n$ to our advantage, we notice that $\sigma_i(\mathbf{e}) = \sigma_n(\mathbf{e}) \sigma_{n-i}(1/\mathbf{e})$. Moreover, $\sigma_n(\mathbf{e}) = \prod_{i=1}^{n} e_i$ can be computed in $\mathcal{O}(n)$. Therefore we introduce the polynomial:

$$Q \triangleq \sigma_n(\mathbf{e})(X + \frac{1}{e_1})(X + \frac{1}{e_2})...(X + \frac{1}{e_n}). \qquad (13)$$

Then if we expand Q to $\beta_0 + \beta_1 X + ... + \beta_n X^n$, we obtain with Vieta's formula again:

$$\forall i \in [\![0, n]\!], \quad \beta_i = \sigma_n(\mathbf{e})\sigma_{n-i}(1/\mathbf{e}) = \sigma_i(\mathbf{e}). \tag{14}$$

Subsequently, in order to compute $\sigma_k(\mathbf{e})$, we only require the k first coefficients of Q, which is very efficient when k is small in comparison with n. This results in a time complexity of $\mathcal{O}(kn)$ (Proposition 5 in Appendix B). Moreover, there are only $\mathcal{O}(\log(n))$ levels of recursion, and since every level can have its operations parallelized, the resulting algorithm scales very well with n when implemented on a GPU. The algorithm is described in Algorithm 1: step 2 initializes the polynomials for the divide and conquer method. While the polynomial has not been fully expanded, steps 5-6 merge branches by performing the polynomial multiplications (which can be done in parallel). Step 10 adjusts the coefficients using equation (14). We point out that we could obtain an algorithm with a time complexity of $\mathcal{O}(n \log(k)^2)$ if we were using Fast Fourier Transform for polynomial multiplications in steps 5-6. Since we are interested in the case where k is small (typically 5), such an improvement is negligible.

Algorithm 1 *Forward Pass*

Require: $\mathbf{e} \in (\mathbb{R}_+^*)^n$, $k \in \mathbb{N}^*$

1: $t \leftarrow 0$
2: $P_i^{(t)} \leftarrow (1, 1/e_i)$ for $i \in [\![1, n]\!]$ ▷ Initialize n polynomials to $X + \frac{1}{e_i}$ (encoded by coefficients)
3: $p \leftarrow n$ ▷ Number of polynomials
4: **while** $p > 1$ **do** ▷ Merge branches with polynomial multiplications
5: $\quad P_1^{(t+1)} \leftarrow P_1^{(t)} * P_2^{(t)}$ ▷ Polynomial multiplication up to degree k
...
6: $\quad P_{(p-1)//2}^{(t+1)} \leftarrow P_{p-1}^{(t)} * P_p^{(t)}$ ▷ Polynomial multiplication up to degree k
7: $\quad t \leftarrow t + 1$
8: $\quad p \leftarrow (p-1)//2$ ▷ Update number of polynomials
9: **end while**
10: $P^{(t+1)} \leftarrow P^{(t)} \times \prod_{i=1}^{n} e_i$ ▷ Recover $\sigma_i(\mathbf{e}) = \sigma_{n-i}(1/\mathbf{e})\sigma_n(\mathbf{e})$
11: **return** $P^{(t+1)}$

Obtaining numerical stability in single float precision requires special attention: the use of exponentials with a temperature parameter potentially arbitrarily small is fundamentally unstable. In Appendix C.1, we describe how operating in the log-space and using the sum-log-exp trick alleviates this issue.

4.3 Backward Computation

A side effect of this procedure is that a large number of buffers are allocated for automatic differentiation: for each addition in log-space, we apply $\log$ and $\exp$ operations, each of which needs to store values for the backward pass. This results in a significant amount of time spent on memory allocations, which become the time bottleneck. To avoid this, we exploit the structure of the problem and design a backward algorithm that relies on the results of the forward pass. By avoiding the memory allocations and considerably reducing the number of operations, the backward pass is then sped up by one to two orders of magnitude and becomes negligible in comparison to the forward pass. We describe our efficient backward pass in more details below.

First, we introduce the notation for derivatives:

$$\text{For } j \in [\![1, n]\!], i \geq 1, \quad \delta_j^{(i)} \triangleq \frac{\partial \sigma_i(\mathbf{e})}{\partial e_j}. \tag{15}$$

We now observe that:

$$\delta_j^{(i)} = \sigma_{i-1}(\mathbf{e}_{\setminus j}). \tag{16}$$

In other words, equation (16) states that $\delta_j^{(i)}$, the derivative of $\sigma_i(\mathbf{e})$ with respect to e_j, is the sum of product of all $(i-1)$-tuples that do not include e_j. One way of obtaining $\sigma_{i-1}(\mathbf{e}_{\setminus j})$ is to compute a forward pass for $\mathbf{e}_{\setminus j}$, which we would need to do for every $j \in [\![1, n]\!]$. To avoid such expensive

computations, we remark that $\sigma_i(\mathbf{e})$ can be split into two terms: the ones that contain e_j (which can expressed as $e_j\sigma_{i-1}(\mathbf{e}_{\setminus j})$) and the ones that do not (which are equal to $\sigma_i(\mathbf{e}_{\setminus j})$ by definition). This gives the following relationship:

$$\sigma_i(\mathbf{e}_{\setminus j}) = \sigma_i(\mathbf{e}) - e_j\sigma_{i-1}(\mathbf{e}_{\setminus j}). \tag{17}$$

Simplifying equation (17) using equation (16), we obtain the following induction for the derivatives:

$$\delta_j^{(i)} = \sigma_{i-1}(\mathbf{e}) - e_j\delta_j^{(i-1)}. \tag{18}$$

Since the $(\sigma_i(\mathbf{e}))_{1\leq i\leq k}$ have been computed during the forward pass, we can initialize the induction with $\delta_j^{(1)} = 1$ and iteratively compute the derivatives $\delta_j^{(i)}$ for $i \geq 2$ with equation (18). This is summarized in Algorithm 2.

Algorithm 2 *Backward Pass*

Require: $\mathbf{e}$, $(\sigma_j(\mathbf{e}))_{1\leq j\leq k}$, $k \in \mathbb{N}^*$ ▷ $(\sigma_j(\mathbf{e}))_{1\leq j\leq k}$ have been computed in the forward pass

1: $\delta_i^{(1)} = 1$ for $i \in [\![1, n]\!]$
2: **for** $j \in [\![1, k]\!]$ **do**
3: $\quad\delta_j^{(i)} = \sigma_{i-1}(\mathbf{e}) - e_j\delta_j^{(i-1)}$ for $i \in [\![1, n]\!]$
4: **end for**

Algorithm 2 is subject to numerical instabilities (Proposition C.2 in Appendix C.2). In order to avoid these, one solution is to use equation (16) for each unstable element and therefore require additional forward passes. To avoid this inefficiency, we derive a novel and error-bounded approximation, which simply requires us to compute the additional $(k+1)$-th coefficient during the forward pass. Note that the time complexity of the forward pass remains unchanged since $\mathcal{O}((k+1)C) = \mathcal{O}(kC)$. Once this additional coefficient is obtained from the forward pass, applying our approximation to each unstable element of the backward pass has a constant time complexity (exactly four floating-point operations) instead of a full forward pass of time complexity $\mathcal{O}(kC)$. For details, we refer the reader to Appendix C.2.

5 Experiments

Theoretical results suggest that Cross-Entropy (CE) is an optimal classifier in the limit of infinite data, by accurately approximating the data distribution. In practice, the presence of label noise makes the data distribution more complex to estimate when only a finite number of samples is available. For these reasons, we explore the behavior of CE and $L_{k,\tau}$ when varying the amount of label noise and the training data size. For the former, we introduce label noise in the CIFAR-100 data set (Krizhevsky, 2009), in a way that would not perturb the top-5 error of a perfect classifier. For the latter, we vary the training data size on subsets of the ImageNet data set (Russakovsky et al., 2015).

In all the following experiments, the temperature parameter is fixed throughout the training and is set to one, unless explicitly stated otherwise. This choice is discussed in the next section. The algorithms are implemented in PyTorch and will be made publicly available. Experiments on CIFAR-100 are performed on a single Nvidia Titan Xp card, and the experiments on ImageNet make use of two Nvidia Titan Xp cards.

5.1 CIFAR-100 with noise

Data set. In this experiment, we investigate the impact of label noise on CE and $L_{5,1}$. The CIFAR-100 data set contains 60,000 RGB images, with 50,000 samples for training-validation and 10,000 for testing. There are 20 "coarse" classes, each consisting of 5 "fine" labels. For example, the coarse class "people" is made up of the five fine labels "baby", "boy", "girl", "man" and "woman". In this set of experiments, the images are centered and normalized channel-wise before they are fed to the network. We use the standard data augmentation technique with random horizontal flips and random crops of size 32×32 on the images padded with 4 pixels on each side.

We introduce noise in the labels as follows: with probability p, each fine label is replaced by a fine label from the same coarse class. This new label is chosen at random and may be identical to

the original label. Note that all instances generated by data augmentation from a single image are assigned the same label. The case $p = 0$ corresponds to the original data set without noise, and $p = 1$ to the case where the label is completely random (within the fine labels of the coarse class). With this method, a perfect top-5 classifier would still be able to achieve 100 % accuracy by systematically predicting the five fine labels of the unperturbed coarse label.

Methods. To evaluate our loss functions, we use the architecture DenseNet 40-40 from Huang et al. (2017), and we use the same hyper-parameters and learning rate schedule as in Huang et al. (2017). The temperature parameter is fixed to one. When the level of noise becomes non-negligible, we empirically find that CE suffers from over-fitting and significantly benefits from early stopping – which our loss does not need. Therefore we help the baseline and hold out a validation set of 5,000 images, on which we monitor the accuracy across epochs. Then we use the model with the best top-5 validation accuracy and report its performance on the test set. Results are averaged over three runs with different random seeds.

Table 1: *Testing performance on CIFAR-100 with different levels of label noise. With noisy labels, $L_{5,1}$ consistently outperforms CE on both top-5 and top-1 accuracies, with improvements increasingly significant with the level of noise. For reference, a model making random predictions would obtain 1% top-1 accuracy and 5% top-5 accuracy.*

Noise	Top-1 Accuracy (%)		Top-5 Accuracy (%)	
Level	CE	$L_{5,1}$	CE	$L_{5,1}$
0.0	**76.68**	69.33	**94.34**	94.29
0.2	68.20	**71.30**	87.89	**90.59**
0.4	61.18	**70.02**	83.04	**87.39**
0.6	52.50	**67.97**	79.59	**83.86**
0.8	35.53	**55.85**	74.80	**79.32**
1.0	14.06	**15.28**	67.70	**72.93**

Results. As seen in Table 1, $L_{5,1}$ outperforms CE on the top-5 testing accuracy when the labels are noisy, with an improvement of over 5% in the case $p = 1$. When there is no noise in the labels, CE provides better top-1 performance, as expected. It also obtains a better top-5 accuracy, although by a very small margin. Interestingly, $L_{5,1}$ outperforms CE on the top-1 error when there is noise, although $L_{5,1}$ is not a surrogate for the top-1 error. For $p = 0.8$, $L_{5,1}$ still yields an accuracy of 55.85%, as compared to 35.53% for CE. This suggests that when the provided label is only informative about top-5 predictions (because of noise or ambiguity), it is preferable to use $L_{5,1}$.

5.2 IMAGENET

Data set. As shown in Figure 1, the ImageNet data set presents different forms of ambiguity and noise in the labels. It also has a large number of training samples, which allows us to explore different regimes up to the large-scale setting. Out of the 1.28 million training samples, we use subsets of various sizes and always hold out a balanced validation set of 50,000 images. We then report results on the 50,000 images of the official validation set, which we use as our test set. Images are resized so that their smaller dimension is 256, and they are centered and normalized channel-wise. At training time, we take random crops of 224 $\times$ 224 and randomly flip the images horizontally. At testing time, we use the standard ten-crop procedure (Krizhevsky et al., 2012).

We report results for the following subset sizes of the data: 64k images (5%), 128k images (10%), 320k images (25%), 640k images (50%) and finally the whole data set (1.28M $-$ 50k $=$ 1.23M images for training). Each strict subset has all 1,000 classes and a balanced number of images per class – except for the largest subset, which has a slight unbalance alike the full ImageNet data set.

Methods. In all following experiments, we train a ResNet-18 (He et al., 2016) following the protocol of the ImageNet experiment in Huang et al. (2017). Namely, we optimize the model with Stochastic Gradient Descent with a batch-size of 256, for a total of 90 epochs. We use a momentum of 0.9. The learning rate is initially set to 0.1 and is then divided by ten at epochs 30 and 60. For both methods, training on the whole data set takes about a day and a half (it is only 10% longer with $L_{5,1}$

than with CE). As previously, the validation top-5 accuracy is monitored at every epoch, and we use the model with best top-5 validation accuracy to report its test error.

Probabilities for Multiple Crops. Using multiple crops requires a probability distribution over labels for each crop. Then this probability is averaged over the crops to compute the final prediction. The standard method is to use a softmax activation over the scores. We believe that such an approach is only grounded to make top-1 predictions. The probability of a label $\bar{y}$ being part of the top-5 prediction should be marginalized over all combinations of 5 labels that include $\bar{y}$ as one of their elements. This can be directly computed with our algorithms to evaluate σ_k and its derivative. We refer the reader to Appendix D for details. All the reported results of top-5 error with multiple crops are computed with this method. This provides a systematic boost of at least 0.2% for all loss functions. In fact, it is more beneficial to the CE baseline, by up to 1% in the small data setting.

Table 2: *Top-5 accuracy (%) on ImageNet using training sets of various sizes. Results are reported on the official validation set, which we use as our test set.*

% Data Set	Number of Images	CE	$L_{5,1}$
100%	1.23M	**90.48**	89.56
50%	640k	**87.56**	87.14
25%	320k	82.43	**82.79**
10%	128k	71.39	**72.24**
5%	64k	59.47	**60.81**

Results. The results of Table 2 confirm that $L_{5,1}$ offers better performance on top-5 error than CE when the amount of training data is restricted. As the data set size increases, CE is able to perform better top-5 predictions. In the following section, we investigate the performance of CE and $L_{5,1}$ using the full data set in more detail.

6 Error Analysis

Influence of the Margin. Recall that CE is equivalent to using $L_{1,1}$ without any margin. Therefore, there are two major differences between CE and $L_{5,1}$: (i) they have different margins, and (ii) they are not optimizing for the same value of k in the top-k error. A natural question that arises is whether it is beneficial to add a margin or not. To answer this, we disentangle the influence of the two aforementioned factors by varying them independently. In other words, we compare the performance of $L_{k,1}$ for $k \in \{1, 5\}$, with a margin (denoted by $L_{k,1}(1)$) and without (denoted by $L_{k,1}(0)$) – with this notation, CE corresponds to $L_{1,1}(0)$. We use the same experimental protocol as in the previous section, which yields the following results:

	Top-1 Accuracy (%)	Top-5 Accuracy (%)
CE=$L_{1,1}(0)$	71.69	90.48
$L_{1,1}(1)$	**71.94**	90.42
$L_{5,1}(0)$	63.66	89.27
$L_{5,1}(1)$	63.05	**89.56**

Table 3: *Impact of k and of the margin on the performance of the losses. For $k \in \{1, 5\}$, we report in bold font the surrogate with best top-k accuracy. In both cases, using a margin improves the top-k accuracy being optimized.*

Two useful facts emerge through these experiments. First, for both values of k, we confirm that using a margin increases the top-k error being optimized by the surrogate. In particular, $L_{1,1}$ gives an improvement over the top-1 error of CE. Second, somewhat surprisingly, the top-1 surrogates provide better top-5 accuracy than top-5 surrogates. This raises the question of whether the use of a top-1 surrogate results in better features, which we investigate next.

Quality of features. We use a model pre-trained with CE and learn the classifier layer with $L_{5,1}$, and vice-versa. This results in the matrix presented below:

		Classifier	
		CE	$L_{5,1}$
Features	CE	90.48	89.44
	$L_{5,1}$	90.05	89.56

Table 4: *Top-*5 *accuracy (%) for cross-combinations of classifiers and features.*

In Table 4, we observe that using the features learned from CE does not help the performance of $L_{5,1}$. More surprisingly perhaps, CE performs better than $L_{5,1}$ itself when using features learned with $L_{5,1}$.

Surrogate Approximation. A more careful analysis reveals that the temperature parameter is playing a crucial role in this specific situation. Indeed, when we evaluate the objective function with the loss function l_5, we obtain a numerical value of 0.83 for the model trained with $L_{5,1}$ and 0.73 for the one trained with CE. While $L_{5,\tau}$ would be a perfect approximation of l_5 in the limit $\tau \to 0$, it appears that the temperature of 1 is too high for this instance of problem.

Improved Results based on Error Analysis. Using the above analysis, we wish to adapt the temperature parameter to more accurately optimize for l_5. Since the model trained with CE has obtained a better objective value for l_5, we start with it and further optimize it with $L_{5,\tau}$ while progressively decreasing the temperature τ from 1 to 0 (details in Appendix E.3). This successfully reduces the objective function for l_5 from 0.73 to 0.52, and results in an increase of the testing accuracy from 90.48% to 90.62%. In previous work, similar fine-tuning on ImageNet has been attempted for different top-k surrogates (Lapin et al., 2017). According to the authors, these experiments did not not compare favorably to additional training with cross-entropy. In our setting, however, the training procedure indicates that CE would not benefit from such supplementary training. Nonetheless, fine-tuning with our top-k surrogate results in a performance improvement.

7 Conclusion

This work has introduced a new family of loss functions that allow for direct minimization of the top-k error. We have experimentally shown that non-sparsity is essential for loss functions to work well with deep neural networks. Thanks to a connection to polynomial algebra and a novel approximation, we have shown how to efficiently evaluate the smooth loss and its gradient.

The experiments have shown that our smooth top-5 loss function is more robust to noise and overfitting than cross-entropy when the amount of training data is limited. In the context of large data, we have shown that using a margin improves results for the error being optimized. Furthermore, we have demonstrated that careful optimization of the non-smooth top-k surrogate leads to improved results. Making this optimization more accurate with an annealing of the temperature is an interesting direction for future research.

Our remark about the necessity for smoothing the loss in deep learning is not specific to top-k error, and we hope that this insight will help the design of other loss functions. In particular, structured prediction problems could benefit from smoothed SVM surrogate losses. How to efficiently compute such smooth functions could open interesting research problems.

REFERENCES

Xiaojun Chang, Yao-Liang Yu, and Yi Yang. Robust top-k multiclass SVM for visual category recognition. *International Conference on Knowledge Discovery and Data Mining*, 2017.

Pratik Chaudhari, Anna Choromanska, Stefano Soatto, and Yann LeCun. Entropy-SGD: Biasing gradient descent into wide valleys. *International Conference on Learning Representations*, 2017.

Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (ELUs). *International Conference on Learning Representations*, 2016.

Koby Crammer and Yoram Singer. On the algorithmic implementation of multiclass kernel-based vector machines. *Journal of Machine Learning Research*, 2001.

Yanbo Fan, Siwei Lyu, Yiming Ying, and Bao-Gang Hu. Learning with average top-k loss. *Neural Information Processing Systems*, 2017.

Caglar Gulcehre, Marcin Moczulski, Francesco Visin, and Yoshua Bengio. Mollifying networks. *International Conference on Learning Representations*, 2017.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *Conference on Computer Vision and Pattern Recognition*, 2016.

Gao Huang, Zhuang Liu, Kilian Q Weinberger, and Laurens van der Maaten. Densely connected convolutional networks. *Conference on Computer Vision and Pattern Recognition*, 2017.

Hao Jiang, Stef Graillat, Roberto Barrio, and Canqun Yang. Accurate, validated and fast evaluation of elementary symmetric functions and its application. *Applied Mathematics and Computation*, 2016.

Alex Krizhevsky. *Learning multiple layers of features from tiny images*, 2009.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Neural Information Processing Systems*, 2012.

Maksim Lapin, Matthias Hein, and Bernt Schiele. Top-k multiclass SVM. *Neural Information Processing Systems*, 2015.

Maksim Lapin, Matthias Hein, and Bernt Schiele. Loss functions for top-k error: Analysis and insights. *Conference on Computer Vision and Pattern Recognition*, 2016.

Maksim Lapin, Matthias Hein, and Bernt Schiele. Analysis and optimization of loss functions for multiclass, top-k, and multilabel classification. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2017.

Yuh-Jye Lee and Olvi L Mangasarian. SSVM: A smooth support vector machine for classification. *Computational optimization and Applications*, 2001.

Yuncheng Li, Yale Song, and Jiebo Luo. Improving pairwise ranking for multi-label image classification. *Conference on Computer Vision and Pattern Recognition*, 2017.

PyTorch. `http://pytorch.org/`.

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision*, 2015.

Alexander G. Schwing, Tamir Hazan, Marc Pollefeys, and Raquel Urtasun. Efficient structured prediction with latent variables for general graphical models. *International Conference on Machine Learning*, 2012.

Caixia Yan, Minnan Luo, Huan Liu, Zhihui Li, and Qinghua Zheng. Top-k multi-class svm using multiple features. *Information Sciences*, 2017.

Alan L. Yuille and Anand Rangarajan. The concave-convex procedure (CCCP). *Neural Information Processing Systems*, 2002.

Hao Zheng, Zhanlei Yang, Wenju Liu, Jizhong Liang, and Yanpeng Li. Improving deep neural networks using softplus units. *International Joint Conference on Neural Networks*, 2015.

A SURROGATE LOSSES

In this section, we fix C the number of classes. We let $\tau > 0$ and $k \in \{1, ..., C-1\}$. The results assume that l_k has the same rescaling as in equation (8):

$$l_k(\mathbf{s}, y) = \max\left\{\left(\frac{1}{k}\mathbf{s}_{\setminus y} + \mathbf{1}\right)_{[k]} - \frac{1}{k}s_y, 0\right\}. \tag{19}$$

A.1 REFORMULATION

Proposition 1. *We can equivalently re-write l_k as:*

$$l_k(\mathbf{s}, y) = \max_{\bar{\mathbf{y}} \in \mathcal{Y}^{(k)}}\left\{\Delta_k(\bar{\mathbf{y}}, y) + \frac{1}{k}\sum_{j \in \bar{\mathbf{y}}} s_j\right\} - \max_{\bar{\mathbf{y}} \in \mathcal{Y}_y^{(k)}}\left\{\frac{1}{k}\sum_{j \in \bar{\mathbf{y}}} s_j\right\}. \tag{20}$$

Proof.

$$\begin{aligned}
l_k(\mathbf{s}, y) &= \max\left\{\left(\frac{1}{k}\mathbf{s}_{\setminus y} + \mathbf{1}\right)_{[k]} - \frac{1}{k}s_y, 0\right\}, \\
&= \max\left\{\left(\frac{1}{k}\mathbf{s}_{\setminus y} + \mathbf{1}\right)_{[k]} - \frac{1}{k}s_y, 0\right\} + \left(\frac{1}{k}\sum_{j=1}^{k-1} s_{[j]} + \frac{1}{k}s_y\right) - \left(\frac{1}{k}\sum_{j=1}^{k-1} s_{[j]} + \frac{1}{k}s_y\right), \\
&= \max\left\{\left(\frac{1}{k}\mathbf{s}_{\setminus y} + \mathbf{1}\right)_{[k]} + \frac{1}{k}\sum_{j=1}^{k-1} s_{[j]}, \frac{1}{k}\sum_{j=1}^{k-1} s_{[j]} + \frac{1}{k}s_y\right\} - \left(\frac{1}{k}\sum_{j=1}^{k-1} s_{[j]} + \frac{1}{k}s_y\right), \\
&= \max\left\{\max_{\bar{\mathbf{y}} \in \mathcal{Y}^{(k)} \setminus \mathcal{Y}_y^{(k)}}\left\{1 + \frac{1}{k}\sum_{j \in \bar{\mathbf{y}}} s_j\right\}, \max_{\bar{\mathbf{y}} \in \mathcal{Y}_y^{(k)}}\left\{\frac{1}{k}\sum_{j \in \bar{\mathbf{y}}} s_j\right\}\right\} - \max_{\bar{\mathbf{y}} \in \mathcal{Y}_y^{(k)}}\left\{\frac{1}{k}\sum_{j \in \bar{\mathbf{y}}} s_j\right\}, \\
&= \max_{\bar{\mathbf{y}} \in \mathcal{Y}^{(k)}}\left\{\Delta_k(\bar{\mathbf{y}}, y) + \frac{1}{k}\sum_{j \in \bar{\mathbf{y}}} s_j\right\} - \max_{\bar{\mathbf{y}} \in \mathcal{Y}_y^{(k)}}\left\{\frac{1}{k}\sum_{j \in \bar{\mathbf{y}}} s_j\right\}.
\end{aligned} \tag{21}$$

□

A.2 POINTWISE CONVERGENCE

Lemma 1. *Let $n \geq 2$ and $\mathbf{e} \in \mathbb{R}^n$. Assume that the largest element of $\mathbf{e}$ is greater than its second largest element: $e_{[1]} > e_{[2]}$. Then $\lim_{\substack{\tau \to 0 \\ \tau > 0}} \tau \log\left(\sum_{i=1}^{n} \exp(e_i/\tau)\right) = e_{[1]}$.*

Proof. For simplicity and without loss of generality, we suppose that the elements of $\mathbf{e}$ are sorted in descending order. Then for $i \in \{2, ..n\}$, we have $e_i - e_1 \leq e_2 - e_1 < 0$ by assumption, and thus $\forall\, i \in \{2, ..n\}$, $\lim_{\substack{\tau \to 0 \\ \tau > 0}} \exp((e_i - e_1)/\tau) = 0$. Therefore:

$$\lim_{\substack{\tau \to 0 \\ \tau > 0}} \sum_{i=1}^{n} \exp((e_i - e_1)/\tau) = \sum_{i=1}^{n} \lim_{\substack{\tau \to 0 \\ \tau > 0}} \exp((e_i - e_1)/\tau) = 1. \tag{22}$$

And thus:

$$\lim_{\substack{\tau \to 0 \\ \tau > 0}} \tau \log\left(\sum_{i=1}^{n} \exp((e_i - e_1)/\tau)\right) = 0. \tag{23}$$

The result follows by noting that:

$$\tau \log \left(\sum_{i=1}^{n} \exp(e_i/\tau) \right) = e_1 + \tau \log \left(\sum_{i=1}^{n} \exp((e_i - e_1)/\tau) \right). \tag{24}$$

□

Proposition 2. *Assume that* $s_{[k-1]} > s_{[k]}$ *and that* $s_{[k]} > s_{[k+1]}$ *or* $\frac{1}{k}s_y > 1 + \frac{1}{k}s_{[k]}$. *Then* $\lim_{\substack{\tau \to 0 \\ \tau > 0}} L_{k,\tau}(\mathbf{s}, y) = l_k(\mathbf{s}, y)$.

Proof. From $s_{[k]} > s_{[k+1]}$ or $\frac{1}{k}s_y > 1 + \frac{1}{k}s_{[k]}$, one can see that $\max_{\bar{\mathbf{y}} \in \mathcal{Y}^{(k)}} \left\{ \Delta_k(\bar{\mathbf{y}}, y) + \frac{1}{k} \sum_{j \in \bar{\mathbf{y}}} s_j \right\}$ is a strict maximum. Similarly, from $s_{[k-1]} > s_{[k]}$, we have that $\max_{\bar{\mathbf{y}} \in \mathcal{Y}_y^{(k)}} \left\{ \frac{1}{k} \sum_{j \in \bar{\mathbf{y}}} s_j \right\}$ is a strict maximum. Since $L_{k,\tau}$ can be written as:

$$\begin{aligned} L_{k,\tau}(\mathbf{s}, y) = \tau \log \Bigg[\sum_{\bar{\mathbf{y}} \in \mathcal{Y}^{(k)}} \exp \left(\left(\Delta_k(\bar{\mathbf{y}}, y) + \frac{1}{k} \sum_{j \in \bar{\mathbf{y}}} s_j \right) / \tau \right) \Bigg] \\ - \tau \log \Bigg[\sum_{\bar{\mathbf{y}} \in \mathcal{Y}_y^{(k)}} \exp \left(\left(\frac{1}{k} \sum_{j \in \bar{\mathbf{y}}} s_j \right) / \tau \right) \Bigg], \end{aligned} \tag{25}$$

the result follows by two applications of Lemma 1. □

A.3 Bound on Non-Smooth Function

Proposition 3. $L_{k,\tau}$ *is an upper bound on* l_k *if and only if* $k = 1$.

Proof. Suppose $k = 1$. Let $\mathbf{s} \in \mathbb{R}^C$ and $y \in \mathcal{Y}$. We introduce $y^* = \arg\max_{\bar{y} \in \mathcal{Y}} \Delta_1(\bar{y}, y) + s_{\bar{y}}$. Then we have:

$$\begin{aligned} l_1(\mathbf{s}, y) &= \Delta_1(y^*, y) + s_{y^*} - s_y, \\ &= \tau \log(\exp((\Delta_1(y^*, y) + s_{y^*})/\tau) - \tau \log \exp(s_y/\tau), \\ &\leq \tau \log(\sum_{\bar{y} \in \mathcal{Y}} \exp((\Delta_1(\bar{y}, y) + s_{\bar{y}})/\tau) - \tau \log \exp(s_y/\tau) = L_{1,\tau}(\mathbf{s}, y). \end{aligned} \tag{26}$$

Now suppose $k \geq 2$. We construct an example $(\mathbf{s}, y)$ such that $L_{k,\tau}(\mathbf{s}, y) < l_k(\mathbf{s}, y)$. For simplicity, we set $y = 1$. Then let $s_1 = \alpha$, $s_i = \beta$ for $i \in \{2, ..., k+1\}$ and $s_i = -\infty$ for $i \in \{k+2, ..., C\}$. Assuming infinite values simplifies the analysis, and by continuity of $L_{k,\tau}$ and l_k, the proof will hold for real values sufficiently small.

We further assume that $1 + \frac{1}{k}(\beta - \alpha) > 0$. Then can write $l_k(\mathbf{s}, y)$ as:

$$l_k(\mathbf{s}, y) = 1 + \frac{1}{k}(\beta - \alpha). \tag{27}$$

Exploiting the fact that $\exp(s_i/\tau) = 0$ for $i \geq k+2$, we have:

$$\sum_{\bar{\mathbf{y}} \in \mathcal{Y}^{(k)} \setminus \mathcal{Y}_y^{(k)}} \prod_{j \in \bar{\mathbf{y}}} \exp((1 + s_j)/k\tau) = \exp \left(\frac{1 + \beta}{\tau} \right), \tag{28}$$

And:

$$\sum_{\bar{\mathbf{y}} \in \mathcal{Y}_y^{(k)}} \exp \left(\left(\frac{1}{k} \sum_{j \in \bar{\mathbf{y}}} s_j \right) / \tau \right) = k \exp \left(\frac{\alpha + (k-1)\beta}{k\tau} \right). \tag{29}$$

This allows us to write $L_{k,\tau}$ as:

$$\begin{aligned}
L_{k,\tau}(\mathbf{s}, y) &= \tau \log\left(k \exp\left(\frac{\alpha + (k-1)\beta}{k\tau}\right) + \exp\left(\frac{1+\beta}{\tau}\right)\right) - \tau \log\left(k \exp\left(\frac{\alpha + (k-1)\beta}{k\tau}\right)\right), \\
&= \tau \log\left(1 + \frac{\exp\left(\frac{1+\beta}{\tau}\right)}{k \exp\left(\frac{\alpha+(k-1)\beta}{k\tau}\right)}\right), \\
&= \tau \log\left(1 + \frac{\exp\left(\frac{1}{\tau}\right)}{k \exp\left(\frac{\alpha-\beta}{k\tau}\right)}\right), \\
&= \tau \log\left(1 + \frac{1}{k} \exp\left(\frac{1}{\tau}(1 + \frac{1}{k}(\beta - \alpha))\right)\right).
\end{aligned} \tag{30}$$

We introduce $x = 1 + \frac{1}{k}(\beta - \alpha)$. Then we have:

$$L_{k,\tau}(\mathbf{s}, y) = \tau \log\left(1 + \frac{1}{k} \exp\left(\frac{x}{\tau}\right)\right), \tag{31}$$

And:

$$l_k(\mathbf{s}, y) = x. \tag{32}$$

For any value $x > 0$, we can find $(\alpha, \beta) \in \mathbb{R}^2$ such that $x = 1 + \frac{1}{k}(\beta - \alpha)$ and our hypotheses are all verified. Consequently, we only have to prove that there exists $x > 0$ such that:

$$\Delta(x) \triangleq \tau \log\left(1 + \frac{1}{k} \exp\left(\frac{x}{\tau}\right)\right) - x < 0. \tag{33}$$

We show that $\lim_{x \to \infty} \Delta(x) < 0$, which will conclude the proof.

$$\begin{aligned}
\Delta(x) &= \tau \log\left(1 + \frac{1}{k} \exp\left(\frac{x}{\tau}\right)\right) - x, \\
&= \tau \log\left(1 + \frac{1}{k} \exp\left(\frac{x}{\tau}\right)\right) - \tau \log(\exp(\frac{x}{\tau})), \\
&= \tau \log\left(\exp\left(\frac{-x}{\tau}\right) + \frac{1}{k}\right) \to \tau \log(\frac{1}{k}) < 0 \text{ when } x \to \infty \text{ since } k \geq 2.
\end{aligned} \tag{34}$$

□

A.4 Bound on Prediction Loss

Lemma 2. *Let $(p, q) \in \mathbb{N}^2$ such that $p \leq q$ and $q \geq 1$. Then $\binom{q}{p} \leq \frac{1}{q}\binom{q}{p+1}$.*

Proof.

$$\begin{aligned}
\frac{\binom{q}{p}}{\binom{q}{p+1}} &= \frac{(q-p-1)!(p+1)!}{(q-p)!p!}, \\
&= \frac{(p+1)}{q-p}.
\end{aligned} \tag{35}$$

This is a monotonically increasing function of $p \geq 0$, therefore it is achieving its minimum for $p = 0$:

$$\frac{\binom{q}{p}}{\binom{q}{p+1}} = \frac{(p+1)}{q-p} \leq \frac{1}{q}. \tag{36}$$

□

Lemma 3. *Assume that $y \notin P_k(\mathbf{s})$. Then we have:*

$$\frac{1}{k} \sum_{\bar{\mathbf{y}} \in \mathcal{Y}_y^{(k)}} \exp\left(\sum_{j \in \bar{\mathbf{y}}} \frac{s_j}{k\tau}\right) \leq \sum_{\bar{\mathbf{y}} \in \mathcal{Y}^{(k)} \backslash \mathcal{Y}_y^{(k)}} \exp\left(\sum_{j \in \bar{\mathbf{y}}} \frac{s_j}{k\tau}\right). \tag{37}$$

Proof. The idea of the proof is to show that we can upper bound the score of elements of $\mathcal{Y}_y^{(k)}$ by elements of $\mathcal{Y}^{(k)} \backslash \mathcal{Y}_y^{(k)}$ by replacing the ground truth score by one of the k largest scores (since $y \notin P_k(\mathbf{s})$). There are not enough elements in a single instance of $\mathcal{Y}^{(k)} \backslash \mathcal{Y}_y^{(k)}$ to do that for every element of $\mathcal{Y}_y^{(k)}$, but Lemma 2 bounds the amount of repetitions of $\mathcal{Y}^{(k)} \backslash \mathcal{Y}_y^{(k)}$ needed.

□

Proposition 4. *$L_{k,\tau}$ is, up to a scaling factor, an upper bound on the prediction loss Λ_k:*

$$L_{k,\tau}(\mathbf{s}, y) \geq (1 - \tau \log(k)) \Lambda_k(\mathbf{s}, y). \tag{38}$$

Proof. Suppose that $\Lambda_k(\mathbf{s}, y) = 0$. Then the inequality if trivial because $L_{k,\tau}(\mathbf{s}, y) \geq 0$.

We now assume that $\Lambda_k(\mathbf{s}, y) = 1$. Then there exist at least k higher scores than s_y. To simplify indexing, we introduce $\mathcal{Z}_y^{(k)} = \mathcal{Y}^{(k)} \backslash \mathcal{Y}_y^{(k)}$ and $\mathcal{T}_k$ the set of k labels corresponding to the k-largest scores. By assumption, $y \notin \mathcal{T}_k$ since y is misclassified. We then write:

$$\sum_{\bar{\mathbf{y}} \in \mathcal{Y}^{(k)}} \exp\left(\Delta(\bar{\mathbf{y}}, y)/\tau\right) \prod_{j \in \bar{\mathbf{y}}} u_j = \exp\left(1/\tau\right) \sum_{\bar{\mathbf{y}} \in \mathcal{Z}_y^{(k)}} \prod_{j \in \bar{\mathbf{y}}} u_j + \sum_{\bar{\mathbf{y}} \in \mathcal{Y}_y^{(k)}} \prod_{j \in \bar{\mathbf{y}}} u_j. \tag{39}$$

Thanks to Lemma 3, we have:

$$\sum_{\bar{\mathbf{y}} \in \mathcal{Z}_y^{(k)}} \prod_{j \in \bar{\mathbf{y}}} u_j \geq \frac{1}{k} \sum_{\bar{\mathbf{y}} \in \mathcal{Y}_y^{(k)}} \prod_{j \in \bar{\mathbf{y}}} u_j. \tag{40}$$

Injecting this back into (49):

$$\sum_{\bar{\mathbf{y}} \in \mathcal{Y}^{(k)}} \exp\left(\Delta(\bar{\mathbf{y}}, y)/\tau\right) \prod_{j \in \bar{\mathbf{y}}} u_j \geq (1 + \frac{1}{k} \exp\left(1/\tau\right)) \sum_{\bar{\mathbf{y}} \in \mathcal{Y}_y^{(k)}} \prod_{j \in \bar{\mathbf{y}}} u_j, \tag{41}$$

And back to the original loss:

$$\begin{aligned} L_{k,\tau}(\mathbf{s}, y) &\geq \tau \log\left[(1 + \frac{1}{k} \exp\left(1/\tau\right)) \sum_{\bar{\mathbf{y}} \in \mathcal{Y}_y^{(k)}} \prod_{j \in \bar{\mathbf{y}}} u_j\right] - \tau \log\left[\sum_{\bar{\mathbf{y}} \in \mathcal{Y}_y^{(k)}} \prod_{j \in \bar{\mathbf{y}}} u_j\right], \\ &= \tau \log(1 + \frac{1}{k} \exp\left(1/\tau\right)) \geq \tau \log(\frac{1}{k} \exp\left(1/\tau\right)) = \tau \log(\frac{1}{k}) + 1 = 1 - \tau \log(k). \end{aligned} \tag{42}$$

□

B TIME COMPLEXITY

Lemma 4. *Let P and Q be two polynomials of degree p and q. The time complexity of obtaining the first r coefficients of PQ is $\mathcal{O}(\min\{r, p\} \min\{r, q\})$.*

Proof. The multiplication of two polynomials can be written as the convolution of their coefficients, which can be truncated at degree r for each polynomial. □

Proposition 5. *The time complexity of Algorithm 1 is $\mathcal{O}(kC)$.*

Proof. Let $N = \log_2(C)$, or equivalently $C = 2^N$. With the divide-and-conquer algorithm, the complexity of computing the k first coefficients of P can be written as:

$$T(k, C) = 2T(k, \frac{C}{2}) + \min\{k, C\}^2. \tag{43}$$

Indeed we decompose $P = Q_1 Q_2$, with each Q_i of degree $C/2$, and for these we compute their k first coefficients in $T(\frac{C}{2})$. Then given the k first coefficients of Q_1 and Q_2, the k first coefficients of P are computed in $\mathcal{O}(\min\{k, C\}^2)$ by Lemma 4. Then we can write:

$$\begin{aligned} T(k, C) &= 2T\big(k, \frac{C}{2}\big) + \min\{k, C\}^2, \\ 2T\big(k, \frac{C}{2}\big) &= 4T\big(k, \frac{C}{4}\big) + 2\min\Big\{k, \frac{C}{2}\Big\}^2, \\ &\dots \\ 2^{N-1}T\big(k, \frac{C}{2^{N-1}}\big) &= \underbrace{2^N T(k, 1)}_{2^N\mathcal{O}(1)=\mathcal{O}(C)} + 2^{N-1}\min\Big\{k, \frac{C}{2^{N-1}}\Big\}^2. \end{aligned} \tag{44}$$

By summing these terms, we obtain $T(k, C) = 2^N T(k, 1) + \sum_{j=0}^{N-1} 2^j \min\Big\{k, \frac{C}{2^j}\Big\}^2$.

Let $n_0 \in \mathbb{N}$ such that $\frac{C}{2^{n_0+1}} < k \leq \frac{C}{2^{n_0}}$. In loose notation, we have $k\frac{2^{n_0}}{C} = \mathcal{O}(1)$. Then we can write:

$$\begin{aligned} \sum_{j=0}^{N-1} 2^j \min\Big\{k, \frac{C}{2^j}\Big\}^2 &= \sum_{j=0}^{n_0} 2^j \min\Big\{k, \frac{C}{2^j}\Big\}^2 + \sum_{j=n_0+1}^{N-1} 2^j \min\Big\{k, \frac{C}{2^j}\Big\}^2, \\ &= \sum_{j=0}^{n_0} 2^j k^2 + \sum_{j=n_0+1}^{N-1} 2^j \left(\frac{C}{2^j}\right)^2, \\ &= (2^{n_0+1} - 1)k^2 + C^2(2^{-n_0-1} - 2^{-N}), \\ &= \mathcal{O}(kC). \end{aligned} \tag{45}$$

Thus finally:

$$\begin{aligned} T(k, C) &= 2^N T(k, 1) + \sum_{j=0}^{N-1} 2^j \min\Big\{k, \frac{C}{2^j}\Big\}^2, \\ &= \mathcal{O}(C) + \mathcal{O}(kC), \\ &= \mathcal{O}(kC). \end{aligned} \tag{46}$$

□

C Numerical Stability

C.1 Forward Pass

In order to ensure numerical stability of the computation, we maintain all computations in the log space: for a multiplication $\exp(x_1)\exp(x_2)$, we actually compute and store $x_1 + x_2$; for an addition $\exp(x_1) + \exp(x_2)$ we use the "log-sum-exp" trick: we compute $m = \max\{x_1, x_2\}$, and store $m + \log(\exp(x_1 - m) + \exp(x_2 - m))$, which guarantees stability of the result. These two operations suffice to describe the forward pass.

C.2 Backward Pass

Proposition 6. *The backward recursion of Algorithm 2 is unstable when $e_j \gg 1$ and $e_j \gg \max_{p \neq j}\{e_p\}$.*

Proof. To see that, assume that when we compute $(\sum_{p=1}^{n} e_p) - e_j$, we make a numerical error in the order of ϵ (e.g $\epsilon \simeq 10^{-5}$ for single-precision floats). With the numerical errors, we obtain approximate $\hat{\delta}$ as follows:

$$\begin{aligned}
&\hat{\delta}_j^{(1)} = 1, \\
&\hat{\delta}_j^{(2)} = \sigma_1(\mathbf{e}) - e_j \hat{\delta}_j^{(1)} = \sum_{p=1}^{n} e_p - e_j = \delta_j^{(2)} + \mathcal{O}(\epsilon), \\
&\hat{\delta}_j^{(3)} = \sigma_2(\mathbf{e}) - e_j \hat{\delta}_j^{(2)} = \sigma_2(\mathbf{e}) - e_j(\delta_j^{(2)} + \mathcal{O}(\epsilon)) = \delta_j^{(3)} + \mathcal{O}(e_j \epsilon)), \\
&\quad ... \\
&\hat{\delta}_j^{(k)} = \sigma_k(\mathbf{e}) - e_j \hat{\delta}_j^{(k-1)} = ... = \delta_j^{(k)} + \mathcal{O}(e_j^{k-1} \epsilon)).
\end{aligned} \tag{47}$$

Since $e_j \gg 1$, we quickly obtain unstable results. □

Proposition 7. *We introduce the following approximation to the gradient:*

$$\tilde{\delta}_j^{(k)} \triangleq \frac{\sigma_k(\mathbf{e})}{e_j} - \frac{\sigma_{k+1}(\mathbf{e})}{e_j^2}. \tag{48}$$

Computing $\tilde{\delta}_j^{(k)}$ with equation (56) is numerically stable, and requires four floating point operations.

Proposition 8. *If the following hypothesis is verified:*

$$\exists\, \epsilon \in (0,1) \ : \quad \frac{\sigma_{k+1}(\mathbf{e})}{e_j \sigma_k(\mathbf{e})} < \epsilon, \tag{49}$$

then the relative error of using the approximation of Proposition (7) can be bounded by:

$$\left| \frac{\delta_j^{(k)} - \tilde{\delta}_j^{(k)}}{\delta_j^{(k)}} \right| < \frac{\epsilon}{1-\epsilon} \qquad (= \epsilon + o(\epsilon) \textit{ when } \epsilon \to 0). \tag{50}$$

Intuition To begin with, let us see why hypothesis (57) makes sense. this hypothesis means that the terms that do not contain e_j are negligible in comparison to the other ones. Assume that $\forall p \neq j, \frac{e_p}{e_j} = \mathcal{O}(\xi)$, with $\xi = o(1)$ and $e_j \xi = \mathcal{O}(1)$. This is the case of the instability shown before. Then we can write:

$$\begin{aligned}
\frac{\sigma_{k+1}(\mathbf{e})}{e_j \sigma_k(\mathbf{e})} &= \frac{\sigma_{k+1}(\mathbf{e})}{e_j \sigma_k(\mathbf{e})}, \\
&= \frac{\sigma_{k+1}(\mathbf{e})/e_j^{k+1}}{\sigma_k(\mathbf{e})/e_j^{k}}, \\
&= \frac{\mathcal{O}(\xi^{k+1})}{\mathcal{O}(\xi^{k})}, \\
&= \mathcal{O}(\xi), \\
&= o(1).
\end{aligned} \tag{51}$$

Subsequently, hypothesis (57) is satisfied in the instability case presented previously. Furthermore, we can show with similar reasoning that $\frac{\sigma_k(\mathbf{e})}{e_j} = \mathcal{O}(1)$ and $\frac{\sigma_{k+1}(\mathbf{e})}{e_j^2} = \mathcal{O}(\xi)$, therefore the approximated gradient $\frac{\sigma_k(\mathbf{e})}{e_j} - \frac{\sigma_{k+1}(\mathbf{e})}{e_j^2}$ is stable to compute. This means that we simply compute one more coefficient ($\sigma_{k+1}(\mathbf{e})$) during the forward pass, in order to obtain a fast and stable backward pass.

Proof. We now prove that equation (56) is true when assuming hypothesis (57). First, we note that:

$$\begin{aligned}
\frac{\sigma_{k+1}(\mathbf{e}_{\setminus j})}{e_j} &< \frac{e_j \sigma_k(\mathbf{e}_{\setminus j}) + \sigma_{k+1}(\mathbf{e}_{\setminus j})}{e_j} \qquad \text{(because every } e_p > 0), \\
&= \frac{\sigma_{k+1}(\mathbf{e})}{e_j}, \\
&< \sigma_k(\mathbf{e})\epsilon \qquad \text{(by assumption of 57).}
\end{aligned} \tag{52}$$

Then we can write:

$$\begin{aligned}\delta_j^{(k)} &= \sigma_{k-1}(\mathbf{e}_{\setminus j}),\\ &= \frac{1}{e_j}\left(\sigma_k(\mathbf{e}) - \sigma_k(\mathbf{e}_{\setminus j})\right),\\ &= \frac{1}{e_j}\left(\sigma_k(\mathbf{e}) - \frac{1}{e_j}\left(\sigma_{k+1}(\mathbf{e}) - \sigma_{k+1}(\mathbf{e}_{\setminus j})\right)\right),\\ &= \frac{\sigma_k(\mathbf{e})}{e_j} - \frac{\sigma_{k+1}(\mathbf{e})}{e_j^2} + \frac{\sigma_{k+1}(\mathbf{e}_{\setminus j})}{e_j^2}.\end{aligned} \tag{53}$$

Therefore we have:

$$\begin{aligned}\left|\delta_j^{(k)} - \left(\frac{\sigma_k(\mathbf{e})}{e_j} - \frac{\sigma_{k+1}(\mathbf{e})}{e_j^2}\right)\right| &= \frac{\sigma_{k+1}(\mathbf{e}_{\setminus j})}{e_j^2},\\ &< \frac{\sigma_k(\mathbf{e})}{e_j}\epsilon \quad \text{(by equation (60))}.\end{aligned} \tag{54}$$

And finally:

$$\begin{aligned}\left|\frac{\delta_j^{(k)} - \left(\frac{\sigma_k(\mathbf{e})}{e_j} - \frac{\sigma_{k+1}(\mathbf{e})}{e_j^2}\right)}{\delta_j^{(k)}}\right| &< \frac{\sigma_k(\mathbf{e})}{e_j\delta_j^{(k)}}\epsilon,\\ &= \frac{\sigma_k(\mathbf{e})}{\sigma_k(\mathbf{e}) - \sigma_k(\mathbf{e}_{\setminus j})}\epsilon,\\ &= \frac{1}{1 - \frac{\sigma_k(\mathbf{e}_{\setminus j})}{\sigma_k(\mathbf{e})}}\epsilon.\end{aligned} \tag{55}$$

Since $x \mapsto \frac{1}{1-x}$ is strictly increasing on [0, 1) and :

$$0 \leq \frac{\sigma_k(\mathbf{e}_{\setminus j})}{\sigma_k(\mathbf{e})} = \frac{\sigma_k(\mathbf{e}_{\setminus j})e_j}{\sigma_k(\mathbf{e})e_j} < \frac{\sigma_{k+1}(\mathbf{e})}{\sigma_k(\mathbf{e})e_j} < \epsilon < 1, \tag{56}$$

we have:

$$\frac{1}{1 - \frac{\sigma_k(\mathbf{e}_{\setminus j})}{\sigma_k(\mathbf{e})}} < \frac{1}{1-\epsilon}, \tag{57}$$

and thus:

$$\left|\frac{\delta_j^{(k)} - \left(\frac{\sigma_k(\mathbf{e})}{e_j} - \frac{\sigma_{k+1}(\mathbf{e})}{e_j^2}\right)}{\delta_j^{(k)}}\right| < \frac{\epsilon}{1-\epsilon}. \tag{58}$$

□

D PROBABILITY FOR TOP-K PREDICTION

We consider the probability of label i being part of the final top-k prediction. To that end, we marginalize over all k-tuples that contain i as one of their element. Then the probability of selecting label i for the top-k prediction can be written as:

$$p_i^{(k)} \propto \sum_{\bar{\mathbf{y}} \in \mathcal{Y}_i^{(k)}} \exp(\sum_{j \in \bar{\mathbf{y}}} s_j). \tag{59}$$

Proposition 9. *The unnormalized probability can be computed as:*

$$p_i^{(k)} \propto \frac{d \log \sigma_i(\exp(\mathbf{s}))}{ds_i}. \tag{60}$$

Proof.

$$\begin{aligned} p_i^{(k)} &\propto \exp(s_i)\sigma_{k-1}(\exp(\mathbf{s}_{\setminus i})), \\ &= \exp(s_i)\frac{d\sigma_i(\exp(\mathbf{s}))}{d\exp(s_i)}, \\ &= \frac{d\sigma_i(\exp(\mathbf{s}))}{ds_i}. \end{aligned} \tag{61}$$

Finally we can rescale the unnormalized probability by $\sigma_k(\exp(\mathbf{s}))$ since it is independent of i. We obtain:

$$\hat{p_i}^{(k)} \propto \frac{1}{\sigma_k(\exp(\mathbf{s}))}\frac{d\sigma_i(\exp(\mathbf{s}))}{ds_i} = \frac{d \log \sigma_i(\exp(\mathbf{s}))}{ds_i}. \tag{62}$$

□

NB. We prefer to use $\frac{d \log \sigma_i(\exp(\mathbf{s}))}{ds_i}$ rather than $\frac{d\sigma_i(\exp(\mathbf{s}))}{ds_i}$ for stability reasons. Once the unnormalized probabilities are computed, they can be normalized by simply dividing by their sum.

E EXPERIMENTS: SUPPLEMENTARY DETAILS

E.1 CIFAR-100

In the main paper, we report the average of the scores for clarity purposes. Here, we also detail the standard deviation of the scores for completeness.

Table 5: *Testing performance on CIFAR-100 with different levels of label noise. We indicate the mean and standard deviation (in parenthesis) for each score.*

Noise	Top-1 Accuracy (%)		Top-5 Accuracy (%)	
Level	CE	$L_{5,1}$	CE	$L_{5,1}$
0.0	**76.68** (0.38)	69.33 (0.27)	**94.34** (0.09)	94.29 (0.10)
0.2	68.20 (0.50)	**71.30** (0.79)	87.89 (0.08)	**90.59** (0.08)
0.4	61.18 (0.97)	**70.02** (0.40)	83.04 (0.38)	**87.39** (0.23)
0.6	52.50 (0.27)	**67.97** (0.51)	79.59 (0.36)	**83.86** (0.39)
0.8	35.53 (0.79)	**55.85** (0.80)	74.80 (0.15)	**79.32** (0.25)
1.0	14.06 (0.13)	**15.28** (0.39)	67.70 (0.16)	**72.93** (0.25)

E.2 FEATURES TRANSFER

The fully connected layer has its parameters reset to random values, while all other layers keep their values from the pre-trained model (and remain fixed throughout the training). The learning rate is initialized at 0.1, and is divided by ten every 10 epochs. The whole training has a budget of 40 epochs. We report the testing performance of the model with best top-5 validation accuracy.

E.3 ANNEALING OF THE TEMPERATURE

The temperature successively takes the values 1, 0.1, 0.01, 0.001 and 0 (which corresponds to the non-smooth loss). For each value of the temperature, the learning rate is set to 0.01 for 10 epochs, then 0.001 for 5 epochs and finally 0.001 for 5 epochs. We report the testing performance of the model with best top-5 validation accuracy.

E.4 EVALUATION OF THE OBJECTIVE FUNCTION

We compare the value of the objective function for different models. To avoid any randomness, this is evaluated on the whole training set, where we take a single centered crop for each image.