Analysis of loss functions for fast single-class classification

Gil Keren[1] **Sivan Sabato**[2] · **Björn Schuller**[1,3]

Abstract
We consider neural network training, in applications in which there are many possible classes, but at test time, the task is a binary classification task of determining whether the given example belongs to a specific class. We define the single logit classification (SLC) task: training the network so that at test time, it would be possible to accurately identify whether the example belongs to a given class in a computationally efficient manner, based only on the output logit for this class. We propose a natural principle, the Principle of Logit Separation, as a guideline for choosing and designing losses suitable for the SLC task. We show that the cross-entropy loss function is not aligned with the Principle of Logit Separation. In contrast, there are known loss functions, as well as novel batch loss functions that we propose, which are aligned with this principle. Our experiments show that indeed in almost all cases, losses that are aligned with the Principle of Logit Separation obtain at least 20% relative accuracy improvement in the SLC task compared to losses that are not aligned with it, and sometimes considerably more. Furthermore, we show that fast SLC does not cause any drop in binary classification accuracy, compared to standard classification in which all logits are computed, and yields a speedup which grows with the number of classes.

1 Introduction

With the advent of Big Data, classifiers can learn fine-grained distinctions and are used for classification in settings with very large numbers of classes. Datasets with up to hundreds of thousands of classes are already in use in the industry [1,2], and such classification tasks have been studied in several works (e.g., [3,4]). Classification with a large number of classes appears naturally in vision, in language modeling and in machine translation [5–7].

When using neural network classifiers, one implication of a large number of classes is a high computational burden at test time. Indeed, in standard neural networks using a softmax

Gil Keren
gil.keren@informatik.uni-augsburg.de

1 ZD.B Chair of Embedded Intelligence for Health Care and Wellbeing, University of Augsburg, Augsburg, Germany

2 Department of Computer Science, Ben-Gurion University of the Negev, Beer-Sheva, Israel

3 GLAM – Group on Language, Audio & Music Imperial College, London, UK

layer and the cross-entropy loss, the computation needed for finding the logits of the classes (the pre-normalized outputs of the top network layer) is linear in the number of classes [8] and can be prohibitively slow for high-load systems, such as search engines and real-time machine-translation systems.

In many applications, the task at test time is not full classification of each example into one of the many possible classes. Instead, the task, each time the trained classifier is used, is to identify whether the current example should be classified into one of a small subset of the possible classes, or even a single class. This class can be different every time the classifier is used. Consider, for example, the case of real-time image search [9,10] from a live feed from multiple cameras. When the user queries for images of object A, the classifier has to process a large number of images, and decide whether each image contains an instance of object A or not. The classifier is then activated for the second time, this time with a query to find images of object B. New images are now processed by the classifier, to determine which ones contain an instance of object B.

The setting described above has various applications in which the object to detect might change every time the model is used. For instance, consider cameras installed on autonomous cars, security cameras for detecting objects of interest, or live face recognition, where a different person is to be identified each time, based on the given query.

In the setting that we consider, while every use of the classifier at test-time tests for a single class (or a small number of classes), the classifier itself must support queries on *any* of the classes, since it will be used again and again, each time with a different class as a query. As the number of classes may be large, it is not reasonable to train a separate model for every possible class that might be queried at test time. Instead, our goal is to have a single model which supports *all* possible class-queries.

For this type of applications, one would ideally like to have a test-time computation that does not depend on the total number of possible classes. A natural approach is to calculate only the logit of the class of interest, and use this value alone to infer whether this is the true class of the example. However, the logit of a single class might only be meaningful in comparison with logits of other classes, in which case, unless the other logits are also computed, it cannot be used to successfully determine whether the test example belongs to the class of interest. We name the goal of inferring class correctness from the logit of that class alone single logit classification (SLC). Note that SLC is a binary classification task, stressing the fact that only one logit is computed. In Fig. 1 we demonstrate the speedup yielded by SLC, compared to binary classification in the method which computes all logits and uses them for normalization, as the number of classes increases. For instance, computing only a single logit yields a 10× speedup in evaluation time when there are 400,000 possible classes. The speedup increases with the number of possible classes. See Sect. 6 for full details on the experimental setting and the resulting speedup.

In this work, we show that when using the standard cross-entropy loss for training, the value of a single logit is not informative enough for determining whether this is indeed the true class for the example. In other words, the cross-entropy loss yields poor accuracy in the SLC task. Further, we identify a simple principle that we name the Principle of Logit Separation. This principle captures an essential property that a loss function must have in order to yield good accuracy in the SLC task. The principle states that to succeed in the SLC task, the training objective should optimize for the following property:

> The value of any logit that belongs to the correct class of any training example should be larger than the value of any logit that belongs to a wrong class of any (same or other) training example.

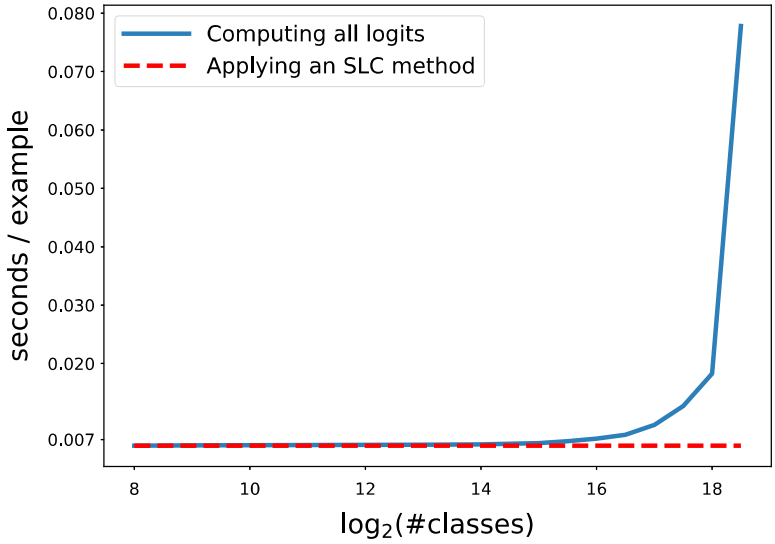

Fig. 1 Computation time of the logits from network inputs, using an inception-V3 image classification architecture [11], where the topmost layer is replaced according to the appropriate number of classes. When applying SLC, computation cost is fixed regardless of the number of classes, which can lead to considerable speedups when the number of classes is large. See Sect. 6 for full details regarding speedups and the experimental setting

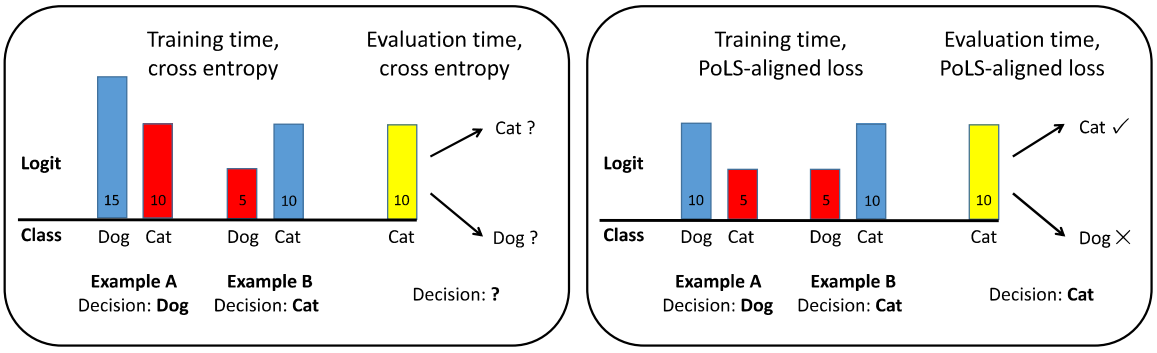

Fig. 2 The Principle of Logit Separation. Left: when training with the cross-entropy loss, the logit values for the class 'Cat' can be the same for two examples, one where it is the true class (blue) and one where it is not (red). Therefore, at test time, a logit with the same value for the class 'Cat' does not indicate whether the example belongs to this class. Right: with a loss function that is aligned with the Principle of Logit Separation, all true logits are greater than all false logits at training time. Hence, at test time, a single logit can indicate the correctness of its respective class (color figure online)

We give a formal definition of the Principle of Logit Separation in Sect. 2. See Fig. 2 for an illustration. We study previously suggested loss functions and their alignment with the Principle of Logit Separation. We show that the Principle of Logit Separation is satisfied by the self-normalization [12] and noise-contrastive estimation [13] training objectives, proposed for calculating posterior distributions in the context of natural language processing, as well as by the binary cross-entropy loss used in multilabel settings [14,15] and the sigmoid Tanimoto loss [16] that is inspired by a distance measure for binary vectors. In contrast, the principle is not satisfied by the standard cross-entropy loss, the max-margin loss, the softmax Tanimoto loss [16], the sigmoid Cauchy–Schwartz divergence and the softmax Cauchy–Schwarz divergence [16,17]. We derive new training objectives for the SLC task based on the Principle of Logit Separation. These objectives are novel batch versions of the cross-entropy loss and the max-margin loss, and we show that they are aligned with the Principle of Logit Separation. In total, we study eleven different training objectives. This work extends the previously published work in [18]. `Tensorflow` code for optimizing the new batch losses is publicly available at https://github.com/EIHW/Logit_Separation.

We corroborate in experiments that the Principle of Logit Separation indeed explains the difference in accuracy of the different loss functions in the SLC task, concluding that training with a loss function that is aligned with the Principle of Logit Separation results in logits that are more informative as a standalone value, and as a result, considerably better SLC accuracy. Specifically, objectives that satisfy the Principle of Logit Separation outperform standard objectives such as the cross-entropy loss on the SLC task with at least 20% relative accuracy improvement in almost all cases, and sometimes considerably more. In another set of experiments, we show that when using loss functions that are aligned with the Principle of Logit Separation, SLC does not cause any decrease in binary classification accuracy for a given class, compared to the case where *all* logits are computed, while keeping the computation cost independent of the total number of classes. Finally, we perform further experiments to determine the speedup factor one can gain by using SLC instead of computing all logits. We show that when the number of classes is large, considerable speedups are gained.

We conclude that by designing a training objective according to the Principle of Logit Separation and applying SLC when the number of classes is large, one can gain considerable speedups at test time without any degradation in model accuracy. As the number of classes in standard datasets has been rapidly growing over the last few years, SLC and the Principle of Logit Separation may play a key role in many applications.

1.1 Related work

We review existing methods that are relevant for faster test-time classification. The hierarchical softmax layer [19] replaces the flat softmax layer with a binary tree with classes as leaves, making the computational complexity of calculating the posterior probability of each class logarithmic in the number of classes. A drawback of this method is the additional construction of the binary tree of classes, which requires expert knowledge or data-driven methods. Inspired by the hierarchical softmax approach, [8] exploit unbalanced word distributions to form clusters that explicitly minimize the average time for computing the posterior probabilities over the classes. The authors report an impressive speedup factor of between 2 and 10 for posterior probability computation, but their computation time still depends on the total number of classes. Differentiated softmax was introduced in [20] as a less computationally expensive alternative to the standard softmax mechanism, in the context of neural language models. With differentiated softmax, each class (word) is represented in the last hidden layer using a different dimensionality, with higher dimensions for more frequent classes. This allows a faster computation for less frequent classes. However, this method is applicable only for highly unbalanced class distributions. Several sampling-based approaches were developed in the context of language modeling, with the goal of approximating the softmax function at training time. Notable examples are importance sampling [21], negative sampling [22], and noise-contrastive estimation (NCE) [13,23]. These methods do not necessarily improve the test-time computational burden; however, we show below that the NCE loss can be used for the SLC task.

2 The Principle of Logit Separation

In the SLC task, the only information about an example is the output logit of the model for the single class of interest. Therefore, a natural approach to classifying whether the class matches the example is to set a threshold: if the logit is above the threshold, classify the

example as belonging to this class, otherwise, classify it as not belonging to the class. We refer to logits that belong to the true classes of their respective training examples as *true logits* and to other logits as *false logits*. For the threshold approach to work well, the values of all true logits should be larger than the value of all false logits across the training sample (in fact, it is enough to separate true and false logits on a class level, but we stick to the stronger assumption in this work). This is illustrated in Fig. 2. The Principle of Logit Separation (PoLS), which was stated in words in Sect. 1, captures this requirement. We formalize this principle below.

Let $[k] := \{1, \ldots, k\}$ be the possible class labels. Assume that the training sample is $S = ((x_1, y_1), \ldots, (x_n, y_n))$, where $x_i \in \mathbb{R}^d$ are the training examples, and $y_i \in [k]$ are the labels of these examples. For a neural network model parametrized by θ, we denote by $z_y^\theta(x)$ the value of the logit assigned by the model to example x for class y. The Principle of Logit Separation (PoLS) can be formally stated as follows:

Definition 1 (*The Principle of Logit Separation*) The Principle of Logit Separation holds for a labeled set S and a model θ, if for any $(x, y), (x', y') \in S$ (including the case $x = x', y = y'$) and any $y'' \neq y'$, we have $z_y^\theta(x) > z_{y''}^\theta(x')$.

The definition assures that every true logit $z_y^\theta(x)$ is larger than every false logit $z_{y''}^\theta(x')$. If this simple principle holds for all train and test examples, it guarantees perfect accuracy in the SLC task, since all true logits are larger than all false logits. Thus, a good approach for a training objective for SLC is to attempt to optimize for this principle on the training set. For a loss ℓ, $\ell(S, \theta)$ is the value of the loss on the training sample using model θ. A loss ℓ is aligned with the Principle of Logit Separation if for any training sample S, a small enough value of $\ell(S, \theta)$ ensures that the requirement in Definition 1 is satisfied for the model θ. In the following sections we study the alignment with the PoLS of known losses and new losses.

3 Objectives that do not satisfy the PoLS

In this section we show that the cross-entropy loss [24], which is the standard loss function for neural network classifiers (e.g., [25]), and the multiclass max-margin loss [26] do not satisfy the PoLS.

3.1 The cross-entropy loss

The cross-entropy loss on a single example is defined as

$$\ell(z, y) = -\log(p_y), \tag{1}$$

where

$$p_y := \frac{e^{z_y}}{\sum_{j=1}^k e^{z_j}} = \left(\sum_{j=1}^k e^{z_j - z_y} \right)^{-1}.$$

Note that p_y is the probability assigned by the softmax layer. It is easy to see that the cross-entropy loss does not satisfy the PoLS. Indeed, as the loss depends only on the difference between logits for every example separately, minimizing it guarantees a certain difference between the true and false logits for every example separately, but does not guarantee that all

true logits are larger than all false logits in the training set. Formally, the following counter-example shows that this loss is not aligned with the PoLS. Let $S = ((x_1, 1), (x_2, 2))$ be the training sample, and let θ_α, for $\alpha > 0$, be a model such that $z^{\theta_\alpha}(x_1) = (2\alpha, \alpha)$, and $z^{\theta_\alpha}(x_2) = (-2\alpha, -\alpha)$. Then, $\ell(S_{\theta_\alpha}) = 2\log(1 + e^{-\alpha})$. Therefore, for any $\epsilon > 0$, there is some $\alpha > 0$ such that $\ell(S_{\theta_\alpha}) \leq \epsilon$, but $z_2^{\theta_\alpha}(x_1) > z_2^{\theta_\alpha}(x_2)$, contradicting an alignment with PoLS.

3.2 The max-margin loss

Max-margin training objectives, most widely known for their role in training support vector machines, are used in some cases for training neural networks [16,27]. Here we consider the multiclass max-margin loss suggested by Crammer and Singer [26], defined as

$$\ell(z, y) = \max(0, \gamma - z_y + \max_{j \neq y} z_j), \tag{2}$$

where $\gamma > 0$ is a hyperparameter that controls the separation margin between the true logit and the false logits of the example. It is easy to see that this loss too does not satisfy the PoLS, since minimizing it again guarantees only a certain difference between the true and false logits for every example separately, and not across the entire training sample. Indeed, consider the same training sample S as defined in the counter-example for the cross-entropy loss above, and the model θ_α defined there. Setting $\alpha = \gamma$, we have $\ell(S_{\theta_\gamma}) = 0$. Thus for any $\epsilon > 0$, $\ell(S_{\theta_\gamma}) < \epsilon$, but $z_2^{\theta_\gamma}(x_1) > z_2^{\theta_\gamma}(x_2)$, contradicting an alignment with PoLS.

3.3 Softmax Cauchy–Schwarz divergence

The Cauchy–Schwarz divergence is based on the Cauchy–Schwarz inequality for inner products, and was defined in [28] for two non-negative real vectors:

$$D_{\mathrm{CS}}(f, g) = -\log \frac{f \cdot g}{||f||\,||g||}, \tag{3}$$

where $\cdot$ denotes an inner product and $||\cdot||$ is the L2 norm. The Cauchy–Schwarz divergence can be viewed as first computing the cosine similarity between the vectors f and g, which results in a value $0 \leq \cos\gamma \leq 1$, where γ is the angle between f and g. For two vectors in the same direction $\cos\gamma = 1$, while $\cos\gamma = 0$ for two orthogonal vectors. Then, a negative logarithm is applied, similarly to the cross-entropy loss, to transform the result to range from 0 for vectors in the same direction to ∞ for orthogonal vectors, which makes it a suitable training objective for matching the vectors f and g.

The Cauchy–Schwarz divergence was used as a classification loss function in [16,17]. For neural network multiclass classifiers, [16] view f and g as vectors of class probabilities, where f is the target probability vector that assigns a probability of 1 to the correct class and 0 to other classes, and g is the model output. As mentioned in [16], the model output class probability vector g can be obtained by applying the sigmoid function $\sigma(z) = (1 + e^{-z})^{-1}$ to every logit, or by applying the softmax normalization function on the logits vector.

This gives rise to two separate loss functions, the softmax Cauchy–Schwarz divergence and the sigmoid Cauchy–Schwarz divergence. We analyze the alignment with the PoLS for each loss separately. The softmax Cauchy–Schwarz divergence is given by

$$\ell(z, y) = -\log \frac{p \cdot t}{||p||\,||t||},$$

where $t_y = 1$ and $t_j = 0$ for $j \neq y$, and p is the model output distribution given by the softmax function

$$p_i = \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}} = \left(\sum_{j=1}^{k} e^{z_j - z_i} \right)^{-1}.$$

Since t is centered in the coordinate y, the loss term can be simplified to

$$\ell(z, y) = -\log \frac{p_y}{||p||}, \tag{4}$$

that is defined for every output probability distribution p, since the softmax function never assigns a probability of 0 to any class.

The softmax Cauchy–Schwarz divergence is not aligned with the PoLS. This can be seen using a counter-example similar to the one used for the cross-entropy loss. Let $S = ((x_1, 1), (x_2, 2))$ be the training sample, and let θ_α, for $\alpha > 0$, be a model such that $z^{\theta_\alpha}(x_1) = (2\alpha, \alpha)$, and $z^{\theta_\alpha}(x_2) = (-2\alpha, -\alpha)$. Further, from the definition of the softmax function, $p^{\theta_\alpha}(x_1) = p^{\theta_\alpha}(x_2)$ therefore also $\ell(z^{\theta_\alpha}(x_1), 1) = \ell(z^{\theta_\alpha}(x_2), 2)$. Since $\ell(S_{\theta_\alpha})$ converges to 0 when α converges to ∞, for every $\epsilon > 0$ there exists $\alpha > 0$ such that $\ell(S_{\theta_\alpha}) \leq \epsilon$, but $z_2^{\theta_\alpha}(x_1) > z_2^{\theta_\alpha}(x_2)$, contradicting an alignment with PoLS.

3.4 Sigmoid Cauchy–Schwarz divergence

When the model output class probability vector g is obtained by applying the sigmoid function $\sigma(z) = (1 + e^{-z})^{-1}$ to the output logits, it gives rise to the sigmoid Cauchy–Schwarz divergence loss function.

The sigmoid Cauchy–Schwarz divergence is given by

$$\ell(z, y) = -\log \frac{\sigma(z) \cdot t}{||\sigma(z)||\,||t||},$$

where $\cdot$ denotes an inner product, t is the same as for the softmax Cauchy–Schwarz divergence, and σ is applied coordinate-wise. Since t is centered in the coordinate y, the loss term can be simplified to

$$\ell(z, y) = -\log \frac{\sigma(z_y)}{||\sigma(z)||}, \tag{5}$$

that is defined for every output logit vector z, since the sigmoid function does not assign a probability of 0 to any class.

The sigmoid Cauchy–Schwarz divergence is not aligned with the PoLS, since it depends only on the ratio between $\sigma(z_y)$ and σ of other coordinates in the logit vector. Formally, the following counter-example shows that this loss is not aligned with the PoLS. Let $S = ((x_1, 1), (x_2, 1))$ be the training sample, and let θ_α, for $\alpha > 0$, be a model such that $\sigma(z^{\theta_\alpha}(x_1)) = (0.5, 0.5/\alpha)$, and $z^{\theta_\alpha}(x_2) = (0.5/\alpha, 0.5/\alpha^2)$. It is easy to derive that $\ell(z^{\theta_\alpha}(x_1), 1) = \ell(z^{\theta_\alpha}(x_2), 1) = -\log \frac{\alpha}{\sqrt{\alpha^2+1}}$. As the last expression converges to 0 when α converges to ∞, for every $\epsilon > 0$ there exists $\alpha > 0$ such that $\ell(S_{\theta_\alpha}) \leq \epsilon$, but $z_2^{\theta_\alpha}(x_1) \not< z_1^{\theta_\alpha}(x_2)$, contradicting an alignment with PoLS.

3.5 Softmax Tanimoto loss

The Tanimoto loss was proposed in [16], based on the Tanimoto distance that is a distance measure for binary vectors [29]. The Tanimoto loss is defined as

$$D_{\text{Tan}}(f, g) = -\frac{f \cdot g}{||f|| + ||g|| - f \cdot g}, \tag{6}$$

where $\cdot$ is an inner product and $||\cdot||$ is the L2 norm. Similarly to the Cauchy–Schwarz divergence, f and g are viewed as vectors of class probabilities, where f is the target probability vector that assigns a probability of 1 to the correct class and 0 to other classes, and g is the model output. Also for this loss, [16] mention that the model output class probability vector g can be obtained by using the softmax or the sigmoid functions. This gives rise to two different loss functions.

When using the softmax function to obtain class probabilities, since $f_y = 1$ and $f_j = 0$ if $j \neq y$, the Tanimoto loss amounts to

$$\ell(z, y) = -\frac{p_y}{1 + ||p|| - p_y} = -\frac{1}{\frac{1+||p||}{p_y} - 1}, \tag{7}$$

where p and p_y are as defined in Sect. 3.3. Since $||p|| \geq p_y$, it follows that $\frac{1+||p||}{p_y} \geq 2$. As a result, $\frac{1+||p||}{p_y} = 2$ if and only if $p_y = 1$. Therefore, the minimal value of ℓ is -1, which is non-standard for loss functions, which are normally defined as non-negative functions. However, for gradient computation purposes it does not matter, and this loss can be used without any changes for training neural network classifiers.

The softmax Tanimoto loss is not aligned with the PoLS, again because the softmax function depends only on the difference between logits. Formally, consider a counter-example similar to the one used for the cross-entropy loss. Let $S = ((x_1, 1), (x_2, 2))$ be the training sample, and let θ_α, for $\alpha > 0$, be a model such that $z^{\theta_\alpha}(x_1) = (2\alpha, \alpha)$, and $z^{\theta_\alpha}(x_2) = (-2\alpha, -\alpha)$. Further, from the definition of the softmax function, $p^{\theta_\alpha}(x_1) = p^{\theta_\alpha}(x_2)$. Therefore, also $\ell(z^{\theta_\alpha}(x_1), 1) = \ell(z^{\theta_\alpha}(x_2), 2)$. Since $\ell(S_{\theta_\alpha})$ converges to -1 when α converges to ∞, for every $\epsilon > -1$ there exists $\alpha > 0$ such that $\ell(S_{\theta_\alpha}) \leq \epsilon$, but $z_2^{\theta_\alpha}(x_1) > z_1^{\theta_\alpha}(x_2)$, contradicting an alignment with the PoLS. We find that the sigmoid Tanimoto loss is aligned with the PoLS, and we discuss this loss function in Sect. 4.4

4 Objectives that satisfy the PoLS

In this section we consider objectives that have been previously suggested for addressing problems that are somewhat related to the SLC task. We show that these objectives indeed satisfy the PoLS.

4.1 Self-normalization

Self-normalization [12] was introduced in the context of neural language models, to avoid the costly step of computing the posterior probability distribution over the entire vocabulary when evaluating the trained models. The self-normalization loss is a sum of the cross-entropy loss with an additional term. Let $\alpha > 0$ be a hyperparameter, and p_y as defined in Eq. (1). The self-normalization loss is defined by

$$\ell(z, y) = -\log(p_y) + \alpha \cdot \log^2\left(\sum_{j=1}^{k} e^{z_j}\right).$$

The motivation for this loss is self-normalization: the second term is minimal when the softmax normalization term $\sum_{j=1}^{k} e^{z_j}$ is equal to 1. When it is equal to 1, the exponentiated logit e^{z_j} can be interpreted as the probability that the true class for the example is j. Devlin et al. [12] report a speedup by a factor of 15 in evaluating models trained when using this loss, since the self-normalization enables computing the posterior probabilities for only a subset of the vocabulary.

Intuitively, this loss should also be useful for the SLC task: if the softmax normalization term is always close to 1, there should be no need to compute it; thus, only the logit of the class in question should be required to infer whether this class is the correct one for the example. Indeed, we show that the self-normalization loss is aligned with the PoLS. When the first term in the loss is minimized for an example, correct and wrong logits are as different as possible from one another. When the second term is minimized for an example, the sum of exponent logits is equal to one. Therefore, when both terms are minimized for an example, the correct logit converges to zero while wrong logits converge to negative infinity. When this is done for the whole training sample, all correct logits are larger than all wrong logits in the training sample. A formal proof is provided in "Appendix A".

4.2 Noise-contrastive estimation

Noise-contrastive estimation (NCE) [13,23] was considered, like self-normalization, in the context of natural language learning. This approach was developed to speed up neural language model training over large vocabularies. In NCE, the multiclass classification problem is treated as a set of binary classification problems, one for each class. Each binary problem classifies, given a context and a word, whether this word is from the data distribution or from a noise distribution. Using only t words from the noise distribution (where t is an integer hyperparameter) instead of the entire vocabulary leads to a significant speedup at training time. Similarly to the self-normalization objective, NCE, in the version appearing in [13], is known to produce a self-normalized logit vector [30]. This property makes NCE a good candidate for the SLC task, as single logit values are informative for the class correctness, and not only when compared to other logits in the same example.

The loss function used in NCE for a single training example, as given by Mnih and Teh [13], is defined based on a distribution over the possible classes, denoted by $q = (q(1), \ldots, q(k))$, where $\sum_{i=1}^{k} q(i) = 1$. The NCE loss, in our notation, is

$$\ell(z, y) = -\log g_y - t \cdot \mathbb{E}_{j \sim q}\left[\log(1 - g_j)\right], \tag{8}$$

where

$$g_j := (1 + t \cdot q(j) \cdot e^{-z_j})^{-1}$$

During training, the second term in the loss is usually approximated by Monte Carlo approximation, using t random samples of $j \sim q$, to speed up training time [13].

We observe that NCE loss is aligned with the PoLS. First, observe that g_j is of a similar form to $\sigma(z_j)$ where $\sigma(z) = (1 + e^{-z})^{-1}$ is the sigmoid function. Therefore, it is easy to see that when the term above is minimized for one example, the value of true logit z_y converges to infinity, and the values of all false logits converge to negative infinity. When the above term

is minimized for the entire training set, all true logits are larger than all false logits across the training set. A formal proof is provided in "Appendix B".

4.3 Binary cross-entropy

The last known loss that we consider is often used in multilabel classification settings. In multilabel settings, each example can belong to several classes, and the goal is to identify the set of classes an example belongs to. A common approach [14,15] is to try to solve k binary classification problems of the form "Does x belong to class j?" using a single neural network model, by minimizing the sum of the cross-entropy losses that correspond to these binary problems. In this setting, the label of each example is a binary vector $(r_1, \ldots, r_k)$, where $r_j = 1$ if x belongs to class j and 0 otherwise. The loss for a single training example with logits z and label-vector r is

$$\ell(z, (r_1, \ldots, r_k)) = -\sum_{j=1}^{n} r_j \log(\sigma(z_j)) + (1 - r_j) \log(1 - \sigma(z_j)),$$

where $\sigma(z) = (1+e^{-z})^{-1}$ is the sigmoid function. This loss can also be used for our setting of multiclass problems, by defining $r_j := \mathbf{1}_{j=y}$ for an example (x, y). This gives the multiclass loss

$$\ell(z, y) = -\log(\sigma(z_y)) + \sum_{j \neq y} \log(1 - \sigma(z_j)).$$

The binary cross-entropy is also aligned with the PoLS. Indeed, similarly to the case of the NCE loss, it is easy to see that when the term above is minimized for one example, the value of true logit z_y converges to infinity, and the values of all false logits converge to negative infinity. When the above term is minimized for the entire training set, all true logits are larger than all false logits across the training set. A formal proof is provided in "Appendix C".

4.4 Sigmoid Tanimoto loss

When using the sigmoid function to obtain class probabilities, the Tanimoto loss from Eq. (6) amounts to

$$\ell(z, y) = -\frac{\sigma(z_y)}{1 + ||\sigma(z)|| - \sigma(z_y)} = -\frac{1}{\frac{1+||\sigma(z)||}{\sigma(z_y)} - 1}, \tag{9}$$

since $f_y = 1$ and $f_j = 0$ for $j \neq y$. Since $||\sigma(z)|| \geq \sigma(z_y)$, it follows that $\frac{1+||\sigma(z)||}{\sigma(z_y)} \geq 2$. As a result, $\frac{1+||\sigma(z)||}{\sigma(z_y)} = 2$ if and only if $\sigma(z_y) = 1$ and $\sigma(z_j) = 0$ for $j \neq y$. Therefore, similarly to the softmax Tanimoto loss, the minimal value of ℓ is -1, which is non-standard but has no practical implications for neural network training.

This loss function is aligned with the PoLS. From the above discussion, if ℓ converges to its minimal value of -1 then $\sigma(z_y)$ converges to 1 and $\sigma(z_j)$ converges to 0 for $j \neq y$. This implies that when ℓ converges to its minimum, z_y converges to ∞ while z_j converges to $-\infty$ for $j \neq y$. Now consider a training sample S and a model θ. It follows that there exists $\epsilon > 0$ such that if $\ell(S_\theta) < -1 + \epsilon$ then for every $(x, y), (x', y') \in S$ and $y'' \neq y'$, we have that $z^\theta_y(x) > 0 > z^\theta_{y''}(x')$, thus this loss is aligned with the PoLS.

5 New training objectives for the SLC task

In this section we propose new training objectives for the SLC task, designed to satisfy the PoLS. These objectives adapt the training objectives of cross-entropy and max-margin, studied in Sect. 3, that do not satisfy the PoLS, by generalizing them to optimize over *batches* of training samples. We show that the revised losses satisfy the PoLS. This approach does not require any new hyperparameters, since the batch size is already a hyperparameter in standard stochastic gradient descent. Further, this allows an easy adaptation of available neural network implementations to the SLC task. When the cross-entropy loss or the max-margin loss are minimized, they guarantee a certain difference between the true and the false logits of each example separately. Our generalization of these losses to batches of examples enforces an ordering also between true and false logits of different examples.

5.1 Batch cross-entropy

Our first batch loss generalizes the cross-entropy loss, which was defined in Eq. (1). The cross-entropy loss can be given as the Kullback–Leibler (KL) divergence between two distributions, as follows. The KL divergence between two discrete probability distributions P and Q over $[k]$ is defined as $\mathrm{KL}(P||Q) := \sum_{i=j}^{k} P(j) \log(P(j)/Q(j))$. For an example (x, y), let $P_{(x,y)}$ be the distribution over $[k]$ which deterministically outputs y, and let Q_x be the distribution defined by the softmax normalized logits, $Q_x(j) = e^{z_j} / \sum_{i=1}^{k} e^{z_i}$. Then it is easy to see that for p_y as defined in Eq. (1), $\mathrm{KL}(P_{(x,y)}||Q_x) = -\log p_y$, exactly the cross-entropy loss in Eq. (1).

We define a batch version of this loss, using the KL divergence between distributions over batches. Recall that the i'th example in a batch B is denoted (x_i, y_i). Let P_B be the distribution over $[m] \times [k]$ defined by

$$P_B(i, j) := \begin{cases} \frac{1}{m} & j = y_i, \\ 0 & \text{otherwise.} \end{cases}$$

Let Q_B be the distribution defined by the softmax normalized logits over the entire batch B. Formally, denote $Z(B) := \sum_{i=1}^{m} \sum_{j=1}^{k} e^{z_j(x_i)}$. Then, $Q_B(i, j) := e^{z_j(x_i)}/Z(B)$. We then define the batch cross-entropy loss as follows.

Definition 2 (*The batch cross-entropy loss*) Let $m > 1$ be an integer, and let B be a uniformly random batch of size m from S. The *batch cross-entropy loss* of a training sample S is

$$\ell(S) := \mathbb{E}_B[L_c(B)], \quad \text{where} \quad L_c(B) := \mathrm{KL}(P_B||Q_B).$$

This batch version of the cross-entropy loss is aligned with the PoLS. Indeed, when this loss is minimized for one training batch, all true logits converge to some positive value (as a normalized exponentiated true logit converges to $1/m$), while all false logits converge to negative infinity (as a normalized exponentiated false logit converges to zero). Therefore, when minimizing this loss across the whole training set, all true logits are larger than all false logits in the training set. A formal proof is provided in "Appendix D".

5.2 Batch max-margin

Our second objective is a batch version of the max-margin loss, which was defined in Eq. (2). For a batch B, denote the minimal true logit in B, and the maximal false logit in B, as follows:

$$z_+^B := \min_{(x,y)\in B} z_y(x), \quad \text{and} \quad z_-^B := \max_{(x,y)\in B, j\neq y} z_j(x).$$

Definition 3 (*The batch max-margin loss*) Let $m > 1$ be an integer, and let B be a uniformly random batch of size m from S. Let ℓ be the single-example max-margin loss defined in Eq. (2), let $\gamma > 0$ be the max-margin hyperparameter. The *batch max-margin* is defined by

$$\ell(S) := \mathbb{E}_B[L_m(B)],$$

where

$$[L_m(B) := \frac{1}{m}\max(0, \gamma - z_+^B + z_-^B) + \frac{1}{m}\sum_{(x,y)\in B} \ell(z(x), y).$$

The batch version of the max-margin loss is aligned with the PoLS. Minimizing the first term in the loss makes sure that all true logits in the batch are larger than all false logits in the batch. Therefore, minimizing the loss over the entire training set makes sure that the PoLS holds. A formal proof is provided in "Appendix D". Note that while the seconds term in the loss is not necessary for ensuring alignment with the PoLS, it is necessary for practical reasons, as without it the gradient is propagated through only two logits from the entire minibatch, which leads to harder optimization and poorer generalization.

6 Experiments

We first empirically show that the PoLS plays a dominant role in SLC accuracy, and that all loss functions that are aligned with the PoLS yield considerably better accuracy in SLC compared to loss functions that are not aligned with the PoLS. We then investigate whether a single logit suffices for obtaining a good binary classification accuracy, compared to the method of computing all logits and using them for normalization. We find that when using a PoLS-aligned loss function, binary classification accuracy is about the same whether we use a single logit only (SLC) or all logits. Lastly, we evaluate the speedups gained by using SLC for binary classification, instead of computing all logits. We show that the speedup increases with the number of classes. For instance, we demonstrate a $10\times$ speedup when the number of classes is approximately 400,000.

6.1 PoLS and SLC accuracy

We tested the SLC tasks on neural networks trained with seven of the objectives we studied. Since the Cauchy–Schwarz divergence and the Tanimoto losses are not as commonly used for training neural network classifiers as the other loss functions we study, we do not include those objectives in our experiments and only include the analysis of their PoLS alignment. To evaluate the success of a learned model in the SLC task we measured, for each class j and each threshold T, the precision and recall in identifying examples from class j using the test $z_j > T$, and calculated the area under the precision–recall curve (AUPRC) defined by

the entire range of possible thresholds. We also measured the precision at fixed recall values (with dictate the threshold T to use) 0.9 (Precision@0.9) and 0.99 (Precision@0.99). We report the averages of these values over all the classes in the dataset.

We tested five computer-vision classification benchmark datasets (using their built-in train/test splits): MNIST [31], SVHN [32] CIFAR-10 and CIFAR-100 [33]. The last dataset is Imagenet [34], which has 1000 classes, demonstrating the scalability of the PoLS approach to many classes. Due to its size, training on Imagenet is highly computationally intensive; therefore, we tested only two representative methods for this dataset, which do not require tuning additional hyperparameters. For every dataset, a single network architecture was used for all training objectives.

The network architectures we used are standard and were fixed before running the experiments. For the MNIST dataset, we used an MLP comprised of two fully connected layers with 500 units each, and an output layer, whose values are the logits, with 10 units. For the SVHN, CIFAR-10 and CIFAR-100 datasets, we used a convolutional neural network [35] with six convolutional layers and one dense layer with 1024 units. The first, third and fifth convolutional layers used a 5×5 kernel, where other convolutional layers used a 1×1 kernel. The first two convolutional layers were comprised of 128 feature maps, where convolutional layers three and four had 256 feature maps, and convolutional layers five and six had 512 feature maps. Max-pooling layers with 3×3 kernel size and a 2×2 stride were applied after the second, fourth and sixth convolutional layers. In all networks, batch normalization [36] was applied to the output of every fully connected or convolutional layer, followed by a rectified-linear nonlinearity. For every combination of a training objective and a dataset (with its fixed network architecture), we optimized for the best learning rate among 1, 0.1, 0.01, 0.001 using the classification accuracy on a validation set. Except for Imagenet, each model was trained for 10^5 steps, which always sufficed for convergence. For the Imagenet experiments, we used an inception-v3 architecture [11] as appears in the `tensorflow` repository. We used all the default hyperparameters from this implementation, changing only the loss function used. For every tested loss function, we trained the inception-v3 model for 6×10^6 iterations.

Experiment results are reported in Table 1. Since many of the measures in our experiments are close to their maximal value of 1, we report the value of *one minus* each measure, so that a smaller number indicates a better accuracy. For each dataset, the losses above the dashed line do not satisfy the PoLS (Sect. 3), while the losses below the line do (Sects. 4 and 5). In the table, the best result for each dataset and measure is indicated in boldface. Finally, the bottom row in each dataset stands for the mean relative improvement between PoLS-aligned losses and other losses, i.e., the relative improvement of the mean of PoLS-aligned losses for a given measure, compared to the mean of losses that are not aligned with the PoLS.

From the results in Table 1, it can be seen that the mean relative improvement of training objectives that are aligned with the PoLS compared to non-aligned objectives is usually at least 20%, and in many cases considerably more. We conclude from these experiments that indeed, alignment with the PoLS is a crucial ingredient for success in the SLC task.

6.2 SLC versus computing all logits

To investigate whether using a single logit (SLC) degrades binary classification accuracy, we compared the binary classification accuracy obtained by SLC with the one obtained by using all logits. In the latter case, we used *all* output logits of the cross-entropy loss after softmax normalization, which is computationally expensive compared to SLC. The experiment setting and binary classification accuracy measures are identical to the one used in Sect. 6.1.

Table 1 Results on single logit classification (SLC), using the different loss functions

Dataset	Method	1-AUPRC	1-Precision@0.9	1-Precision@0.99
MNIST	CE	0.008	0.005	0.203
	max-margin	0.012	0.018	0.262
	self-norm	0.002	0.001	**0.021**
	NCE	0.002	0.002	**0.021**
	binary CE	0.002	**0.000**	0.037
	batch CE	**0.001**	0.001	0.022
	batch max-margin	0.002	0.001	0.034
	Mean relative improvement	82.0%	91.3%	88.4%
SVHN	CE	0.023	0.028	0.545
	max-margin	0.021	0.025	0.532
	self-norm	**0.015**	0.014	0.298
	NCE	0.021	0.017	0.320
	binary CE	**0.015**	0.016	0.312
	batch CE	**0.015**	**0.013**	**0.280**
	batch max-margin	0.018	0.020	0.384
	Mean relative improvement	23.4%	39.6%	40.8%
CIFAR-10	CE	0.109	0.326	0.703
	max-margin	0.094	0.285	0.705
	self-norm	0.073	0.204	0.599
	NCE	0.081	0.214	**0.594**
	binary CE	**0.070**	0.210	0.607
	batch CE	0.072	**0.202**	0.602
	batch max-margin	0.075	0.226	0.636
	Mean relative improvement	26.9%	30.9%	13.6%
CIFAR-100	CE	0.484	0.866	0.974
	max-margin	0.490	0.893	0.977
	self-norm	0.378	0.807	0.970
	NCE	0.383	**0.795**	0.964
	binary CE	0.426	0.870	0.978
	batch CE	**0.371**	**0.795**	**0.961**
	batch max-margin	0.468	0.903	0.983
	Mean relative improvement	16.8%	5.2%	0.5%
Imagenet (1000 classes) ($6 \cdot 10^6$ iterations)	CE	0.366	0.739	0.932
	batch CE	**0.245**	**0.563**	**0.865**
	Relative improvement	33.1%	23.8%	7.2%

Lower values are better. In almost all cases, loss functions that are aligned with the Principle of Logit Separation (under the dashed line) yield a mean relative improvement of at least 20% in SLC accuracy measures, and sometimes considerably more

Results are presented in Table 2. The first two rows present results for binary classification using a single logit (SLC): the first row reports the mean result for loss functions that are aligned with the PoLS reported in Table 1. The second row reports the results for the batch cross-entropy, which is the PoLS-aligned loss that obtained the best accuracy in Table 1. The third row reports results for the cross-entropy method, in which all logits are computed and used for normalization.

The results in Table 2 show that cross-entropy with all logits yields results that are comparable to the results obtained with a single logit, and specifically, with the batch cross-entropy: none of the approaches is consistently more successful in classification than the other. Since calculating cross-entropy with all logits is more computationally demanding, we conclude

Table 2 Comparing binary classification with a single logit (SLC) versus all logits

Dataset	Method	1-AUPRC	1-Precision@0.9	1-Precision@0.99
MNIST	Mean PoLS methods	0.002	0.001	0.027
	batch CE	0.001	0.001	0.022
	CE with all logits	0.001	0.000	0.020
SVHN	Mean PoLS methods	0.017	0.016	0.319
	batch CE	0.015	0.013	0.280
	CE with all logits	0.015	0.016	0.313
CIFAR-10	Mean PoLS methods	0.074	0.211	0.608
	batch CE	0.072	0.202	0.602
	CE with all logits	0.074	0.214	0.648
CIFAR-100	Mean PoLS methods	0.405	0.834	0.971
	batch CE	0.371	0.795	0.961
	CE with all logits	0.380	0.801	0.973
Imagenet	batch CE	0.245	0.563	0.865
	CE with all logits	0.223	0.566	0.872

Lower values are better. PoLS-aligned SLC methods are above the dashed line. Results are comparable; thus, SLC does not cause any degradation in binary classification accuracy, compared to the case where all logits are computed

that in our setting of binary classification with multiple classes at test time, SLC, and specifically batch cross-entropy, is an attractive alternative to standard cross-entropy with all logits.

6.3 SLC speedups

We estimated the speedups gained by performing SLC, compared to methods in which all logits are computed. We used five prominent image classification architectures (Alexnet [25], VGG-16 [37], Inception-v3 [11], Resnet-50 and Resnet-101 [38]). As we are interested in test-time performance, we only measure the time required for computing the forward-pass of a given network. To measure SLC computation time, we replace the top layer by a layer with single unit and measure the time to compute the single logit given an input to the network. To measure the computation time when computing the logits of k classes, we replace the top layer with a layer containing k units, and again measure the time it takes to compute all logits, given an input to the network.

The computation time of a model generally does not depend on its input data or on its accuracy. The time to compute a forward pass of a given model is practically identical whether the input data is random noise or originates from a real dataset of the same dimensions and data range. Therefore, in these experiments we use random noise as input to the networks, and the networks themselves are randomly initialized and not trained. Computation is done using `Tensorflow` and a single NVIDIA Maxwell Titan-X GPU, and forward-pass computation time per example is averaged across 100 minibatches of 32 examples. We use the public implementation of all architectures, as appears in the `tensorflow` repository.

The timing results are given in Table 3. For each network architecture, the first row reports the forward-pass computation time with a single logit. The following rows correspond to different numbers of classes. We report the forward-pass computation time, as well as the speedup obtained by using SLC for this number of classes. This speedup is calculated as the

Table 3 Speedup experiment results

Architecture	Classes	Inference time (s)	SLC speedup
Alexnet	1 (SLC)	3.6×10^{-3}	–
	2^{10}	3.7×10^{-3}	$\times 1.04$
	2^{14}	4.0×10^{-3}	$\times 1.14$
	2^{16}	5.7×10^{-3}	$\times 1.59$
	2^{18}	20.2×10^{-3}	$\times 5.68$
	$2^{18.5}$	76.0×10^{-3}	$\times 21.38$
VGG-16	1 (SLC)	9.4×10^{-3}	–
	2^{10}	9.6×10^{-3}	$\times 1.02$
	2^{14}	9.9×10^{-3}	$\times 1.05$
	2^{16}	11.4×10^{-3}	$\times 1.20$
	2^{18}	26.4×10^{-3}	$\times 2.79$
	$2^{18.5}$	80.5×10^{-3}	$\times 8.52$
Inception-v3	1 (SLC)	6.0×10^{-3}	–
	2^{10}	6.2×10^{-3}	$\times 1.03$
	2^{14}	6.5×10^{-3}	$\times 1.09$
	2^{16}	7.3×10^{-3}	$\times 1.22$
	2^{18}	18.7×10^{-3}	$\times 3.11$
	$2^{18.5}$	76.6×10^{-3}	$\times 12.75$
Resnet-50	1 (SLC)	6.1×10^{-3}	–
	2^{10}	6.4×10^{-3}	$\times 1.04$
	2^{14}	6.6×10^{-3}	$\times 1.08$
	2^{16}	7.4×10^{-3}	$\times 1.20$
	2^{18}	19.1×10^{-3}	$\times 3.11$
	$2^{18.5}$	78.0×10^{-3}	$\times 12.69$
Resnet-101	1 (SLC)	8.0×10^{-3}	–
	2^{10}	8.2×10^{-3}	$\times 1.03$
	2^{14}	8.3×10^{-3}	$\times 1.04$
	2^{16}	9.4×10^{-3}	$\times 1.19$
	2^{18}	23.5×10^{-3}	$\times 2.95$
	$2^{18.5}$	80.4×10^{-3}	$\times 10.10$

When the number of examples is large, SLC results in a considerable speedup

ratio between the computation time for this number of classes and the computation time for SLC (the first row). As expected, the results show a speedup for all architectures, with larger speedups when there are more classes.

For networks with up to $2^{14} = 16{,}384$ classes, the speedup is relatively small, since computation of the network layers other than the logit layer dominates the forward-pass computation time. In contrast, when there are many classes, the computation of logits dominates the forward-pass computation time. Hence, SLC obtains a $\times 2.8 - \times 5.7$ speedup for $2^{18} = 262{,}144$ classes, and $\times 8.5 - \times 21.3$ speedup for $2^{18.5} = 370{,}727$ classes.

In our experiments in Sects. 6.1 and 6.2, we showed that our findings scale well from 10 to 100 and 1000 classes, and we expect these results and findings to scale further to models with a larger number of classes. Ideally, we would have directly tested datasets with hundreds of thousands of classes, to show that the results from Sects. 6.1 and 6.2 scale to datasets with this many classes. However, since such datasets are very large (for instance, the Imagenet-21K dataset has 21,000 classes and more than 14 million examples), these experiments were infeasible with our computational resources. In comparison, a single Inception-V3 model for the significantly smaller Imagenet dataset ($\sim$ 1 million examples) took approximately 3 weeks to train on a single GPU.

We conclude from this set of experiments that using SLC instead of computing all logits can result in considerable speedups that grow with the number of classes.

7 Conclusion

We considered the single logit classification (SLC) task, which is important in various applications. We formulated the Principle of Logit Separation (PoLS) and studied its alignment with eleven loss functions, including the standard cross-entropy loss and two novel loss functions. We established, and corroborated in experiments, that PoLS-aligned loss functions yield more class logits that are more useful for binary classification. We further demonstrated that training with a PoLS-aligned loss function and applying SLC leads to considerable speedups when there are many classes, with no degradation in accuracy. Recent years have seen a constant increase in the number of classes in datasets from various domains; thus, we expect SLC and the PoLS to play a key role in applications.

We note some limitations of this work. First, using SLC instead of full classification to get a speedup at test time is possible only when the number of classes is large. Specifically, our experiments on common image classification networks obtain meaningful speedups when the number of classes is at least 2^{16}. In addition, while the experiments demonstrate the superiority of PoLS-aligned loss functions in the SLC task, and the fact that they scale with the number of classes, the datasets used in the experiments contained only up to 1000 classes, due to limited computational resources. It should further be noted that a speedup at test time requires considering a limited number of classes for each test examples.

In future work, we plan to extend the scope of the Principle of Logit Separation by applying it to other training mechanisms that do not involve loss functions [39] and to neural network regressors designed as classifiers [40,41]. In addition, we plan to formulate the Principle of Logit Separation in a more general framework, that is independent of the choice of model.

Acknowledgements This project has received funding from the European Union's Horizon 2020 research and innovation programme under Grant Agreement No. 826506 (sustAGE). Sivan Sabato was supported in part by the Israel Science Foundation (Grant No. 555/15).

Appendix

Omitted proofs: alignment with the Principle of Logit Separation

All the considered losses are a function of the output logits and the example labels. For a network model θ, denote the vector of logits it assigns to example x by $z^\theta(x) = (z_1^\theta(x), \ldots, z_k^\theta(x))$. When θ and x are clear from context, we write z_j instead of $z_j^\theta(x)$.

Denote the logit output of the sample by $S_\theta = ((z^\theta(x_1), y_1), \ldots, (z^\theta(x_n), y_n))$. A loss function $\ell : \cup_{n=1}^{\infty}(\mathbb{R}^k \times [k])^n \to \mathbb{R}_+$ assigns a loss to a training sample based on the output logits of the model and on the labels of the training examples. The goal of training is to find a model θ which minimizes $\ell(S_\theta) \equiv \ell(S, \theta)$. In almost all the losses we study, the loss on the training sample is the sum over all examples of a loss defined on a single example: $\ell(S_\theta) \equiv \sum_{i=1}^{n} \ell(z^\theta(x_i), y_i)$, thus we only define $\ell(z, y)$. We explicitly define $\ell(S_\theta)$ below only when this is not the case.

A Self-normalization

We prove that the self-normalization loss satisfies the PoLS: let a training sample S and a neural network model θ, and consider an example $(x, y) \in S$. We consider the two terms of the loss in order. First, consider $-\log(p_y)$. From the definition of p_y (Eq. 1) we have that

$$-\log(p_y) = \log\left(\sum_{j=1}^{k} e^{z_j - z_y}\right) = \log\left(1 + \sum_{j \neq y} e^{z_j - z_y}\right).$$

Set $\epsilon_0 := \log(1 + e^{-2})$. Then, if $-\log(p_y) < \epsilon_0$, we have $\sum_{j \neq y} e^{z_j - z_y} \leq e^{-2}$, which implies that (a) $\forall j \neq y, z_j \leq z_y - 2$ and (b) $e^{z_y} \geq \sum_{j=1}^{k} e^{z_j}/(1 + e^{-2}) \geq \frac{1}{2}\sum_{j=1}^{k} e^{z_j}$. Second, consider the second term. There is an $\epsilon_1 > 0$ such that if $\log^2(\sum_{j=1}^{k} e^{z_j}) < \epsilon_1$ then (c) $2e^{-1} < \sum_{j=1}^{k} e^{z_j} < e$, which implies $e^{z_y} < e$ and hence (d) $z_y < 1$. Now, let θ such that $\ell(S_\theta) \leq \epsilon := \min(\epsilon_0, \epsilon_1)$. Then $\forall (x, y) \in S$, $\ell(z^\theta(x), y) \leq \epsilon$. From (b) and (c), $e^{-1} < \frac{1}{2}\sum_{j=1}^{k} e^{z_j} < e^{z_y}$, hence $z_y > -1$. Combining with (d), we get $-1 < z_y < 1$. Combined with (a), we get that for $j \neq y$, $z_j < -1$. To summarize, $\forall (x, y), (x', y') \in S$ and $\forall y'' \neq y'$, we have that $z_y^\theta(x) > -1 > z_{y''}^\theta(x')$, implying PoLS alignment.

B Noise-contrastive estimation

Recall the definition of the NCE loss from Eq. (8):

$$\ell(z, y) = -\log g_y - t \cdot \mathbb{E}_{j \sim q}\left[\log(1 - g_j)\right]$$

where $g_j := (1 + t \cdot q(j) \cdot e^{-z_j})^{-1}$. We prove that the NCE loss satisfies the PoLS: g_j is monotonic increasing in z_j. Hence, if the loss is small, g_y is large and g_j for $j \neq y$, is small. Formally, fix t, and let a training sample S. There is an $\epsilon_0 > 0$ such that if $-\log g_j \leq \epsilon_0$, then $z_j > 0$. Also, there is an $\epsilon_1 > 0$ (which depends on q) such that if $-\mathbb{E}_{j \sim q}\left[\log(1 - g_j)\right] \leq \epsilon_1$ then $\forall j \neq y$, $\log(1 - g_j)$ must be small enough so that $z_j < 0$. Now, consider θ such that $\ell(S_\theta) \leq \epsilon := \min(\epsilon_0, \epsilon_1)$. Then for every $(x, y) \in S$, $\ell(z^\theta(x), y) \leq \epsilon$. This implies that for every $(x, y), (x', y') \in S$ and $y'' \neq y'$, we have that $z_y^\theta(x) > 0 > z_{y''}^\theta(x')$, thus this loss is aligned with the PoLS.

C Binary cross-entropy

This loss is similar in form to the NCE loss: for g_j as in Eq. (8), $g_j = \sigma(z_j - \ln(t \cdot q(j)))$. Since $\sigma(z_j)$ is monotonic, the proof method for NCE carries over and thus the binary cross-entropy loss satisfies the PoLS as well.

D Batch losses

Recall that the batch losses are defined as $\ell(S_\theta) := \mathbb{E}_B[L(B_\theta)]$, where B_θ is a random batch out of S_θ and L is L_c for the cross-entropy (Definition 2), and L_m is the max-margin loss (Definition 3). If true logits are greater than false logits in every batch separately when using, then the PoLS is satisfied on the whole sample, since every pair of examples appears together in some batch. The following lemma formalizes this:

Lemma 1 *If L is aligned with the PoLS, and ℓ is defined by $\ell(S_\theta) := \mathbb{E}_B[L(B_\theta)]$, then ℓ is also aligned with the PoLS.*

Proof Assume a training sample S and a neural network model θ. Since L is aligned with the PoLS, there is some $\epsilon' > 0$ such if $L(B_\theta) < \epsilon'$, then for each $(x, y), (x', y') \in B$ and $y'' \neq y'$ we have that $z^\theta_y(x) > z^\theta_{y''}(x')$. Let $\epsilon = \epsilon' / \binom{n}{m}$, and assume $\ell(S_\theta) < \epsilon$. Since there are $\binom{n}{m}$ batches of size m in S, this implies that for every batch B of size m, $L(B_\theta) \leq \epsilon'$. For any $(x, y), (x', y') \in S$, there is a batch B that includes both examples. Thus, for $y'' \neq y'$, $z^\theta_y(x) > z^\theta_{y''}(x')$. Since this holds for any two examples in S, ℓ is also PoLS-aligned. □

(1) *Batch cross-entropy* To show that the batch cross-entropy satisfies the PoLS, we show that L_c does, which by Lemma 1 implies this for ℓ. By the continuity of KL, and since for discrete distributions, $\mathrm{KL}(P||Q) = 0 \iff P \equiv Q$, there is an $\epsilon > 0$ such that if $L(B_\theta) \equiv \mathrm{KL}(P_B||Q^\theta_B)] < \epsilon$, then for all i, j, $|P_B(i, j) - Q^\theta_B(i, j)| \leq \frac{1}{2m}$. Therefore, for each example $(x, y) \in B$,

$$\frac{e^{z^\theta_y(x)}}{Z(B)} > \frac{1}{2m}, \qquad \text{and} \qquad \forall j \neq y, \quad \frac{e^{z^\theta_j(x)}}{Z(B)} < \frac{1}{2m}.$$

It follows that for any two examples $(x, y), (x', y') \in B$, if $y \neq y'$, then $z^\theta_y(x) > \frac{1}{2m} > z^\theta_{y'}(x')$. Therefore L satisfies the PoLS, which completes the proof.

(2) *Batch max-margin* To show that the batch max-margin loss satisfies the PoLS, we show this for L_m and invoke Lemma 1. Set $\epsilon = \gamma/m$. If $L(B_\theta) < \epsilon$, then $\gamma - z^B_+ + z^B_- < \gamma$, implying $z^B_+ > z^B_-$. Hence, any $(x, y), (x', y') \in B$ such that $y \neq y'$ satisfy $z^\theta_y(x) \geq z^B_+ > z^B_- \geq z^\theta_y(x')$. Thus L is aligned with the PoLS, implying the same for ℓ.

References

1. Deng J, Dong W, Socher R, Li L, Li K, Li F (2009) Imagenet: a large-scale hierarchical image database. In: Proceedings of CVPR, Miami, FL, pp 248–255
2. Partalas I, Kosmopoulos A, Baskiotis N, Artières T, Paliouras G, Gaussie É, Androutsopoulos I, Amini M, Gallinari P (2015) LSHTC: a benchmark for large-scale text classification. arXiv:1503.08581
3. Weston J, Makadia A, Yee H (2013) Label partitioning for sublinear ranking. In: Proceedings of ICML, Atlanta, GA, pp 181–189
4. Gupta MR, Bengio S, Weston J (2014) Training highly multiclass classifiers. J Mach Learn Res 15(1):1461–1492
5. Bahdanau D, Cho K, Bengio Y (2015) Neural machine translation by jointly learning to align and translate. In: Proceedings of ICLR, San Diago, CA
6. Józefowicz R, Vinyals O, Schuster M, Shazeer N, Wu Y (2016) Exploring the limits of language modeling. arXiv:1602.02410
7. Dean TL, Ruzon MA, Segal M, Shlens J, Vijayanarasimhan S, Yagnik J (2013) Fast, accurate detection of 100,000 object classes on a single machine. In: Proceedings of CVPR, Portland, OR, pp 1814–1821
8. Grave E, Joulin A, Cissé M, Grangier D, Jégou H (2017) Efficient softmax approximation for GPUs. In: Proceedings of ICML, Sydney, Australia, pp 1302–1310

9. Maturana D, Scherer S (2015) Voxnet: a 3d convolutional neural network for real-time object recognition. In: Proceedings of IROS, Hamburg, Germany, pp 922–928
10. Redmon J, Divvala SK, Girshick RB, Farhadi A (2016) You only look once: Unified, real-time object detection. In: Proceedings of CVPR, Las Vegas, NV, pp 779–788
11. Szegedy C, Vanhoucke V, Ioffe S, Shlens J, Wojna Z (2016) Rethinking the inception architecture for computer vision. In: Proceddings of CVPR, Las Vegas, NV, pp 2818–2826
12. Devlin J, Zbib R, Huang Z, Lamar T, Schwartz RM, Makhoul J (2014) Fast and robust neural network joint models for statistical machine translation. In: Proceedings of the 52nd annual meeting of the association for computational linguistics. ACL, Baltimore, MD, pp 1370–1380
13. Mnih A, Teh YW (2012) A fast and simple algorithm for training neural probabilistic language models. In: Proceedings ICML, Edinburgh, Scotland, UK
14. Wang J, Yang Y, Mao J, Huang Z, Huang C, Xu W (2016) CNN-RNN: a unified framework for multi-label image classification. In: Proceedings of CVPR, NV, Las Vegas, pp 2285–2294
15. Huang Y, Wang W, Wang L, Tan T (2013) Multi-task deep neural network for multi-label learning. In: Procedding of IEEE international conference on image processing. ICIP, Melbourne, Australia, pp 2897–2900
16. Janocha K, Czarnecki WM (2017) On loss functions for deep neural networks in classification. arXiv:1702.05659
17. Czarnecki WM, Jozefowicz R, Tabor J (2015) Maximum entropy linear manifold for learning discriminative low-dimensional representation. ECML PKDD 2015:52–67
18. Keren G, Sabato S, Schuller B (2018) Fast single-class classification and the principle of logit separation. In: Proceedings of the international conference on data mining (ICDM), Singapore, pp 227–236
19. Morin F, Bengio Y (2005) Hierarchical probabilistic neural network language model. In: Proceedings of AISTATS. Bridgetown, Barbados
20. Chen W, Grangier D, Auli M (2016) Strategies for training large vocabulary neural language models. In: Proceedings of the 54th annual meeting of the association for computational linguistics. ACL, Berlin, Germany
21. Bengio Y, Senecal J (2008) Adaptive importance sampling to accelerate training of a neural probabilistic language model. IEEE Trans Neural Netw 19(4):713–722
22. Mikolov T, Sutskever I, Chen K, Corrado GS, Dean J (2013) Distributed representations of words and phrases and their compositionality. In: Proceedings of NIPS. Lake Tahoe, NV, pp 3111–3119
23. Gutmann M, Hyvärinen A (2010) Noise-contrastive estimation: a new estimation principle for unnormalized statistical models. In: Proceedings of AISTATS. Chia Laguna Resort, Sardinia, Italy, pp 297–304
24. Hinton GE (1989) Connectionist learning procedures. Artif Intell 40(1):185–234
25. Krizhevsky A, Sutskever I, Hinton GE (2012) Imagenet classification with deep convolutional neural networks. In: Proceedings of NIPS. Lake Tahoe, NV, pp 1106–1114
26. Crammer K, Singer Y (2001) On the algorithmic implementation of multiclass kernel-based vector machines. J Mach Learn Res 2:265–292
27. Socher R, Lin CC, Ng AY, Manning CD (2011) Parsing natural scenes and natural language with recursive neural networks. In: Proceedings of ICML. Bellevue, WA, pp 129–136
28. Kampa K, Hasanbelliu E, Príncipe JC (2011) Closed-form Cauchy–Schwarz PDF divergence for mixture of gaussians. IJCNN 2011:2578–2585
29. Gower JC (1985) Measures of similarity, dissimilarity and distance. Encycl Stat Sci 5:397–405
30. Andreas J, When Klein D (2015) why are log-linear models self-normalizing In: Proceddings of NAACL HLT, the 2015 conference of the North American chapter of the association for computational linguistics: human language technologies. Denver, CO, pp. 244–249
31. LeCun Y, Bottou L, Bengio Y, Haffner P (1998) Gradient-based learning applied to document recognition. Proc IEEE 86(11):2278–2324
32. Netzer Y, Wang T, Coates A, Bissacco A, Wu B, Ng AY (2011) Reading digits in natural images with unsupervised feature learning. In: NIPS workshop on deep learning and unsupervised feature learning. Granada, Spain
33. Krizhevsky A (2009) Learning multiple layers of features from tiny images. Master's thesis, Department of Computer Science, University of Toronto
34. Russakovsky O, Deng J, Su H, Krause J, Satheesh S, Ma S, Huang Z, Karpathy A, Khosla A, Bernstein MS, Berg AC, Li F (2015) Imagenet large scale visual recognition challenge. Int J Comput Vis 115(3):211–252
35. LeCun Y, Boser B, Denker JS, Henderson D, Howard RE, Hubbard W, Jackel LD (1989) Backpropagation applied to handwritten zip code recognition. Neural Comput 1(4):541–551
36. Ioffe S, Szegedy C (2015) Batch normalization: accelerating deep network training by reducing internal covariate shift. In: Proceedings of ICML. Lille, France, pp 448–456

37. Simonyan K, Zisserman A (2015) Very deep convolutional networks for large-scale image recognition. In: Proceedings of ICLR. San Diago, CA
38. He K, Zhang X, Ren S, Sun J (2016) Deep residual learning for image recognition. In: Proceedings of CVPR. Las Vegas, NV, pp 770–778
39. Keren G, Sabato S, Schuller B (2017) Tunable sensitivity to large errors in neural network training. In: Proceedings of AAAI, San Francisco, CA, pp 2087–2093
40. Keren G, Cummins N, Schuller BW (2018) Calibrated prediction intervals for neural network regressors. IEEE Access 6:54033–54041
41. Keren G, Han J, Schuller B (2018) Scaling speech enhancement in unseen environments with noise embeddings. In: Proceedings of the CHiME workshop on speech processing in everyday environments. Hyderabad, India, pp 25–29