UniLoss: Unified Surrogate Loss by Adaptive Interpolation

Anonymous authors
Paper under double-blind review

Abstract

We introduce UniLoss, a unified framework to generate surrogate losses for training deep networks with gradient descent, reducing the amount of manual design of task-specific surrogate losses. Our key observation is that in many cases, evaluating a model with a performance metric on a batch of examples can be re-factored into four steps: from input to real-valued scores, from scores to comparisons of pairs of scores, from comparisons to binary variables, and from binary variables to the final performance metric. Using this re-factorization we generate a unified differentiable approximation of the evaluation computation, through adaptive interpolation at selective configurations of the binary variables. Using UniLoss, we can optimize for different tasks and metrics using one unified framework, achieving comparable performance compared with task-specific losses. We validate the effectiveness of UniLoss on three tasks and four datasets.

1 Introduction

Many supervised learning tasks involve designing and optimizing a loss function that is often different from the actual performance metric for evaluating models. For example, cross-entropy is a popular loss function for training a multi-class classifier, but when it comes to comparing models on a test set, classification error is used instead.

Why not optimize the performance metric directly? Because many metrics or output decoders are non-differentiable and cannot be optimized with gradient based methods such as stochastic gradient descent. Non-differentiability occurs when the output of the task is discrete (e.g. class labels), or when the output is continuous but the performance metric has discrete operations (e.g. percentage of real-valued predictions within a certain range of the ground truth).

To address this issue, designing a differentiable loss that serves as a surrogate to the original metric is a standard practice. For standard tasks with simple output and metrics, there exist well-studied surrogate losses. For example, cross-entropy or hinge loss for classification, both of which have proven to work well in practice.

However, designing surrogate losses can sometimes incur substantial manual effort, including a large amount of trial and error and hyper-parameter tuning. For example, a standard evaluation of single-person human pose estimation—predicting the 2D locations of a set of body joints for a single person in an image—involves computing the percentage of predicted body joints that are within a given radius of the ground truth. This performance metric is non-differentiable. Existing work instead trains a deep network to predict a heatmap for each type of body joints, minimizing the difference between the predicted heatmap and a “ground truth” heatmap consisting of a Gaussian bump at the ground truth location (Tompson et al., 2014; Newell et al., 2016). The decision for what error function to use for comparing heatmaps and how to design the “ground truth” heatmaps are manually tuned to optimize performance.

This manual effort in conventional losses is tedious but necessary, because a poorly designed loss can be misaligned with the final performance metric and lead to ineffective training. As we show in the experiment section, without carefully-tuned loss hyper-parameters, conventional manual losses can work poorly.

In this paper we seek to reduce the amount of manual design of surrogate losses by introducing a single surrogate loss applicable to a wide range of tasks. We provide a unified framework to

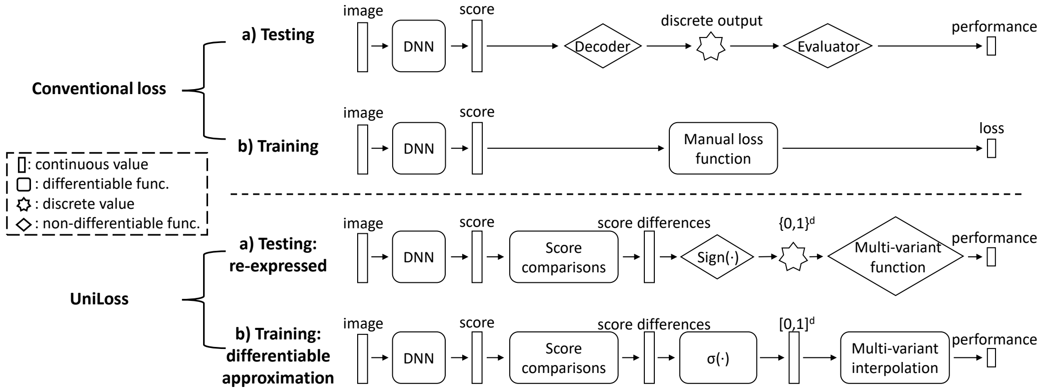

Figure 1: Computation graphs for conventional losses and UniLoss. Top: (a) testing for conventional losses. The decoder and evaluator are usually non-differentiable. (b) training for conventional losses. To avoid the non-differentiability, conventional methods optimize an manually-designed differentiable loss function instead during training. Bottom: (a) re-expressed testing in UniLoss. We re-express the testing so that the non-differentiability exists only in Sign$(\cdot)$ and the multi-variant function. (b) training in UniLoss with differentiable approximation of re-expressed testing. $\sigma(\cdot)$ is the sigmoid function. We approximate the non-differentiable components in the re-expressed testing pipeline with adaptive interpolation methods.

mechanically generate a surrogate loss given a performance metric in the context of deep learning. This means that we only need to specify the performance metric (e.g. classification error) and the inference algorithm—the network architecture, a "decoder" that converts the network output (e.g. continuous scores) to the final output (e.g. discrete class labels), and an "evaluator" that converts the labels to final metric—and the rest is taken care of automatically as part of the training algorithm.

We introduce UniLoss (Fig. 1), a unified framework to generate surrogate losses for training deep networks with gradient descent. We maintain the basic algorithmic structure of mini-batch gradient descent: for each mini-batch, we perform inference on all examples, compute a loss using the results and the ground truths, and generate gradients using the loss to update the network parameters. Our novelty is that we generate the surrogate loss in a unified way for various tasks instead of manually design it for each task.

The key insight behind UniLoss is that for many tasks and performance metrics, evaluating a deep network on a set of training examples—pass the examples through the network, the output decoder, and the evaluator to the performance metric—can be re-expressed into a sequence of four transformations: the training examples are first transformed to a set of real scores, then to some real numbers representing comparisons (through subtractions) of certain pairs of the real-valued scores, then to a set of binary variables, and finally to a single real number. Take multi-class classification as an example, the training examples are first transformed to a set of scores (one per class per example), and then to pairwise comparisons (substractions) between the scores for each example (i.e. the argmax operation), and then to a set of binary values, and finally to a classification accuracy.

The final performance metric is non-differentiable with respect to network weights because the decoder and the evaluator are non-differentiable. But this re-expression allows us to generate differentiable approximation of each non-differentiable transformation through interpolation. Specifically, the transformation from comparisons to binary variables is non-differentiable, we can approximate it by using the sigmoid function to interpolate the sign function. And the transformation from binary variables to final metric may be non-differentiable, we can approximate it by multivariate interpolation—we sample a set of binary input configurations as "anchors", compute the values of final metric at those anchors, and interpolate smoothly between them. Because there are exponentially many possible input configurations, we need to perform *adaptive interpolation*, that is, adaptively selecting a tractable subset based on the current input configuration. We use a mix of three types of anchors, those with high performance globally, those with low performance globally, and those near the current input configuration. We use this strategy to includes both local and global information.

The proposed UniLoss is general and can be applied to various tasks and performance metrics. To validate its effectiveness, we perform experiments on three tasks, binary classification, multi-class classification, and pose estimation. Using UniLoss, we can optimize deep networks for each task in a unified way, achieving comparable performance as manual losses designed for each task.

Our main contributions in this work are: 1) We propose a framework—UniLoss—to generate losses for various tasks without tedious task-specific manual design of surrogate losses. 2) We propose a new general perspective of surrogate losses—evaluation is re-expressed as a sequence of four transformations, where each non-differntiable transformation can be approximated with differentiable functions. 3) We demonstrate the effectiveness of UniLoss on three tasks and four datasets, achieving comparable performance with conventional losses.

2 Related Work

Direct Loss Minimization The line of direct loss minimization works are related to UniLoss because we share a similar idea of finding a good approximation of the performance metric. There have been much efforts to directly minimize specific classes of tasks and metrics. For example, Taylor et al. (2008) optimize ranking metrics such as Normalized Discounted Cumulative Gain by smoothing them with an assumed probabilistic distribution of documents. Henderson & Ferrari (2016) directly optimize mean average precision in object detection by computing "pseudo partial derivatives" for various continuous variables. Nguyen & Sanner (2013) explore to optimize the 0-1 loss in binary classification by search-based methods including branch and bound search, combinatorial search, and also coordinate descent on the relaxations of 0-1 losses. Liu et al. (2016) propose to improve the conventional cross-entropy loss by multiplying a preset constant with the angle in the inner product of the `softmax` function to encourage large margins between classes. In addition to the large algorithmic differences, these works also differ from ours in that they are tightly coupled with specific tasks and applications.

Hazan et al. (2010) and Song et al. (2016) proved that under mild conditions, optimizing a max-margin structured-output loss is asymptotically equivalent to directly optimizing the performance metrics. Specifically, assume a model in the form of a differentiable scoring function $S(x, y; w) : \mathcal{X} \times \mathcal{Y} \rightarrow \mathbf{R}$ that evaluates the compatibility of output y with input x. During inference, they predict the y_w with the best score: $y_w = \text{argmax}_y \ S(x, y; w)$. During training, in addition to this regular inference, they also perform the *loss-augmented inference* (Tsochantaridis et al., 2005; Hazan et al., 2010): $y^\dagger = \text{argmax}_y \ S(x, y; w) + \epsilon\xi(y_w, y)$, where ξ is the final performance metric (in terms of error), and ϵ is a small time-varying weight.

While above max-margin losses can ideally work with many different performance metrics ξ, its main limitation in practical use is that it can be highly nontrivial to design an efficient algorithm the loss-augmented inference, as it often requires some clever factorization of the performance metric ξ over the components of the structured output y. In fact, for many metrics the loss-augmented inference is NP-hard and one must resort to designing approximate algorithms, which further increases the difficulty of practice use.

In contrast, our method does not demand the same level of human ingenuity. The main human effort involves re-factoring the inference code and evaluation code to a particular format, which may be further eliminated by automatic code analysis. There is no need to design a new inference algorithm over discrete outputs and analyze its efficiency. The difficulty of designing loss-augmented inference algorithms for each individual task makes it impractical to compare fairly with max-margin methods on diverse tasks, because it is unclear how to design the inference algorithms.

Surrogate Losses There has been a large body of literature studying surrogate losses, for tasks including multi-class classification (Bartlett et al., 2006; Zhang, 2004; Tewari & Bartlett, 2007; Crammer & Singer, 2001; Allwein et al., 2000; Ramaswamy et al., 2013; 2014), binary classification (Bartlett et al., 2006; Zhang, 2004; Ramaswamy et al., 2013; 2014) and pose estimation (Tompson et al., 2014). Compared to them, UniLoss removes the need for tedious manual effort to design task-specific losses. We demonstrate that UniLoss, as a general loss framework, can be applied for all these tasks and achieve comparable performance.

Reinforcement Learning Reinforcement learning algorithms have been used to optimize performance metrics for structured output problems, especially those that can be (re-)formulated as taking a sequence of actions (Ranzato et al., 2016; Liu et al., 2017; Caicedo & Lazebnik, 2015; Yeung et al., 2016). For example, Liu et al. (2017) use policy gradients (Sutton et al., 2000) to optimize metrics for image captioning. We differ from these approaches in two key aspects. First, we do not need to (re-)formulate a task as sequential decision problem, which may be natural for certain tasks such as text generation, but unnatural for others such as human pose estimation. Second, reinforcement learning approaches treat performance metrics as black boxes, whereas we assume access to the code of the performance metrics, which is a valid assumption in most cases. This access allows us to reason about the code and generate better gradients.

3 UniLoss

3.1 Overview

UniLoss provides a unified way to generate a surrogate loss for training deep networks with mini-batch gradient descent without task-specific design.

Formally, let $\mathbf{x} = (x_1, x_2, \dots, x_n) \in \mathcal{X}^n$ be a set of n training examples and $\mathbf{y} = (y_1, y_2, \dots, y_n) \in \mathcal{Y}^n$ be the ground truth. Let $\phi(\cdot; w) : \mathcal{X} \to \mathbf{R}^d$ be a deep network parameterized by weights w that outputs a d-dimensional vector; let $\delta : \mathbf{R}^d \to \mathcal{Y}$ be a decoder that decodes the network output to a possibly discrete final output; let $\xi : \mathcal{Y}^n \times \mathcal{Y}^n \to \mathbf{R}$ be an evaluator. ϕ and δ are applied in a mini-batch fashion on $\mathbf{x} = (x_1, x_2, \dots, x_n)$; the performance e of the deep network is then

$$e = \xi(\delta(\phi(\mathbf{x}; w)), \mathbf{y}). \tag{1}$$

Our approach seeks to automatically find a surrogate loss to minimize e, with the novel observation that in many cases e can be re-expressed as

$$e = g(h(f(\phi(\mathbf{x}; w), \mathbf{y}))), \tag{2}$$

where $\phi(\cdot; w)$ is the same as in Eqn. 1, representing a deep neural network, $f : \mathbf{R}^{n \times d} \times \mathcal{Y}^n \to \mathbf{R}^l$ is differentiable and maps outputed real numbers and the ground truth to l comparisons each representing the difference between certain pair of real numbers , $h : \mathbf{R}^l \to \{0, 1\}^l$ maps the l score differences to l binary variables, and $g : \{0, 1\}^l \to \mathbf{R}$ computes the performance metric from binary variables. Note that h has a restricted form that always maps continuous values to binary values through sign function, whereas g can be arbitrary computation that maps binary values to a real number.

We give intermediate outputs some notations: 1) training examples $\mathbf{x}, \mathbf{y}$ are transformed to scores $\mathbf{s} = (s_1, s_2, \dots, s_{nd})$, where $\mathbf{s} = \phi(\mathbf{x}; w)$. 2) $\mathbf{s}$ is converted to some comparisons (differences of two scores) $\mathbf{c} = (c_1, c_2, \dots, c_l)$, where $\mathbf{c} = f(\mathbf{s}, \mathbf{y})$. 3) $\mathbf{c}$ is converted to binary variables $\mathbf{b} = (b_1, b_2, \dots, b_l)$ representing the binary outcome of the comparisons, where $\mathbf{b} = h(\mathbf{c})$. 4) the binary variables are transformed to a single real number by $e = g(\mathbf{b})$.

This new re-expression of a performance metric allows us to decompose the metric e with g, h, f and ϕ, where ϕ and f are differentiable functions but h and g are often non-differentiable. The non-differentiability of h and g causes e to be non-differentiable with respect to network weights w.

Our UniLoss generates differentiable approximations of the non-differentiable h and g through interpolation, thus making the metric e optimizable with gradient descent. Formally, UniLoss gives a differentiable approximation $\tilde{e}$

$$\tilde{e} = \tilde{g}(\tilde{h}(f(\phi(\mathbf{x}; w), \mathbf{y}))), \tag{3}$$

where f and ϕ are the same as in Eqn. 2, and $\tilde{h}$ and $\tilde{g}$ are the differentiable approximation of h and g.

3.2 Example: Multi-class Classification

We take multi-class classification as an example to show how re-expression works. First, we give formal definitions of multi-class classification and the performance metric: prediction accuracy.

Inputs are a mini-batch of images $\mathbf{x} = (x_1, x_2, \dots, x_n)$ and their corresponding ground truth labels are $\mathbf{y} = (y_1, y_2, \dots, y_n)$ where n is the batch size. $y_i \in \{1, 2, \dots, p\}$ and p is the number of classes, which happens to be the same value as d in Sec. 3.1. A network $\phi(\cdot;w)$ outputs a score matrix $\mathbf{s}=[s_{i,j}]_{n\times p}$ and $s_{i,j}$ represents the score for the i-th image belongs to the class j.

The decoder $\delta(\mathbf{s})$ decodes $\mathbf{s}$ into the discrete outputs $\tilde{\mathbf{y}}=(\tilde{y}_1,\tilde{y}_2,\ldots,\tilde{y}_n)$ by $\tilde{y}_i=\mathrm{argmax}_{1\leq j\leq p}\ s_{i,j}$, and $\tilde{y}_i$ represents the predicted label of the i-th image for $i=1,2,\ldots,n$. The evaluator $\xi(\tilde{\mathbf{y}},\mathbf{y})$ evaluates the accuracy e from $\tilde{\mathbf{y}}$ and $\mathbf{y}$ by $e=\frac{1}{n}\sum_{i=1}^{n}[y_i=\tilde{y}_i]$, where $[\cdot]$ is the Iverson bracket. Considering above together, the predicted label for an image is correct if and only if the score of its ground truth class is higher than the score of every other class:

$$[y_i=\tilde{y}_i]=\bigwedge_{\substack{1\leq j\leq p\\ j\neq y_i}}[s_{i,y_i}-s_{i,j}>0],\ \text{for all } 1\leq i\leq n, \tag{4}$$

where $\wedge$ is logical and.

We thus re-express the decoding and evaluation process as a sequence of $f(\cdot)$ that transforms $\mathbf{s}$ to comparisons—$s_{i,y_i}-s_{i,j}$ for all $1\leq i\leq n, 1\leq j\leq p$, and $j\neq y_i$ ($n\times(p-1)$ comparisons in total), $h(\cdot)$ that transforms comparisons to binary values using $[\cdot>0]$, and $g(\cdot)$ that transforms binary values to e using logical and.

3.3 Decomposition

In Eqn. 2, after we transform the training examples into scores $\mathbf{s}=(s_1,s_2,\ldots,s_{nd})$, we get the score comparisons (differences of pairs of scores) $\mathbf{c}=(c_1,c_2,\ldots,c_l)$ using $\mathbf{c}=f(\mathbf{s},\mathbf{y})$. Each comparison is $c_i=s_{k_i^1}-s_{k_i^2}$, $1\leq i\leq l, 1\leq k_i^1,k_i^2\leq nd$. The function h then transforms the comparisons to binary values by $\mathbf{b}=h(\mathbf{c})$. The h is the sign function, i.e. $b_i=[c_i>0]$, $1\leq i\leq l$. The function g then computes e by $e=g(\mathbf{b})$, where g can be arbitrary computation that converts binary values to a real number. In practice, g can be complex and vary significantly across tasks and metrics.

This decomposition holds under the assumption that there is no learnable parameter in the evaluation. This assumption is reasonable because many prior works advocate the use of a single network followed by a simple decoder (Newell et al., 2017; Newell & Deng, 2017; Redmon et al., 2016; Redmon & Farhadi, 2017). Actually, given any performance metrics involving discrete operations in function ξ and δ in Eqn. 1 (otherwise the metric e is differentiable and trivial to be handled), the computation of function $\xi(\delta(\cdot))$ can be re-expressed as a sequence of continuous operations (which is optional), discrete operations that make some differentiable real numbers non-differentiable, and any following operations. The discrete operations always occur when there are step functions, which can be expressed as comparing two numbers, to the best of our knowledge.

This decomposition is usually straightforward to obtain from the specification of the decoding and evaluating procedures. The main manual effort is in identifying the discrete comparisons (binary variables). Then we simply write the discrete comparisons as function f and h, and represent its following operations as function g.

In later sections we will show how to identify the binary variables for three commonly-used metrics, which can be trivially extended to other performance metrics. On the other hand, this process is largely a mechanical exercise, as it is equivalent to rewriting some existing code in an alternative rigid format. We expect the binary variables to be automatically generated in future work through automatic analysis of code.

3.4 Interpolation of h

In $\mathbf{b}=h(\mathbf{c})$, each element $b_i=[c_i>0]$. We approximate the step function $[\cdot]$ using the `sigmoid` function. That is, $\tilde{\mathbf{b}}=\tilde{h}(\mathbf{c})=(\tilde{b}_1,\tilde{b}_2,\ldots,\tilde{b}_l)$, and each element

$$\tilde{b}_i=\texttt{sigmoid}(c_i), \tag{5}$$

where $1\leq i\leq l$. We now have $\tilde{h}$ as the differentiable approximation of h.

3.5 Adaptive Interpolation of g

We approximate $g(\cdot)$ in $e=g(\mathbf{b})$ by multivariate interpolation over the input $\mathbf{b}\in\{0,1\}^l$. More specifically, We first sample a set of configurations as "anchors" $\mathbf{a}=(\mathbf{a}_1,\mathbf{a}_2,\ldots,\mathbf{a}_t)$, where $\mathbf{a}_i$ is a configuration of $\mathbf{b}$, and compute the output values $g(\mathbf{a}_1), g(\mathbf{a}_2), \ldots, g(\mathbf{a}_t)$, where $g(\mathbf{a}_i)$ is the actual performance metric value and t is the number of anchors sampled. We then get a interpolated function over the anchors as $\tilde{g}(\cdot; \mathbf{a})$. We finally get $\tilde{e} = \tilde{g}(\tilde{\mathbf{b}}; \mathbf{a})$, where $\tilde{\mathbf{b}}$ is computed from $\tilde{h}$, f and ϕ.

By choosing a differentiable interpolation method, the $\tilde{g}$ function becomes trainable using gradient based methods. We use a common yet effective interpolation method: inverse distance weighting (IDW) (Shepard, 1968):

$$\tilde{g}(\mathbf{u}; \mathbf{a}) = \begin{cases} \frac{\sum_{i=1}^{t} \frac{1}{d(\mathbf{u}, \mathbf{a}_i)} g(\mathbf{a}_i)}{\sum_{i=1}^{t} \frac{1}{d(\mathbf{u}, \mathbf{a}_i)}}, & d(\mathbf{u}, \mathbf{a}_i) \neq 0 \text{ for all } 1 \leq i \leq t; \\ g(\mathbf{a}_i), & d(\mathbf{u}, \mathbf{a}_i) = 0 \text{ for some } i. \end{cases} \tag{6}$$

where $\mathbf{u}$ represents the input to $\tilde{g}$ and $d(\mathbf{u}, \mathbf{a}_i)$ is the Euclidean distance between $\mathbf{u}$ and $\mathbf{a}_i$.

We perform *adaptive interpolation* in UniLoss, by adaptively selecting a subset of anchors based on the current training examples. We use a mix of three types of anchors—good anchors with high performance values globally, bad anchors with low performance values globally, and nearby anchors that are close to the current configuration, which is computed from the current training examples and network weights. By using both the global information from the good and bad anchors and the local information from the nearby anchors, we are able to get an informative interpolation surface.

In general, we obtain good anchors by flipping some bits from the best anchor, which is the configuration derived from the ground truth. The bad anchors are generated by randomly sampling binary variables. The nearby anchors are obtained by flipping some bits from the current configuration.

4 Experimental Results

We first demonstrate UniLoss's ability on non-trivial metrics and tasks: for example, we experiment with average precision (area under the precision-recall curve), a metric that can only be evaluated on a set of images, and involves not only correctness of each image but also the ranking of multiple images. In addition, we also experiment with pose estimation where the output is structured and manual design of loss is not straightforward. We then show that UniLoss also performs similarly as well-established conventional losses on common multi-class classification tasks.

4.1 Average Precision for Binary Classification

Task and Metric Binary classification is to classify an example from two classes—positives and negatives. Potential applications include face classification and image retrieval. Here we use the handwritten digit dataset, MNIST, as an example. Given a 28 × 28 image containing a single number from 0 to 9, we classify it into the zero (positive) class or the non-zero (negative) class. MNIST has 60k training images and 10k test images with the positive-negative ratio of 1:9. We create a validation split by reserving 6k images from the original training set. On this unbalanced binary classification, average precision (AP), known as area under the precision-recall curve, is the performance metric.

Implementation Details The output of a mini-batch of n images is $\mathbf{s} = (s_1, s_2, \ldots, s_n)$, where s_i represents the predicted score of i-th image to be positive. The evaluation process is essentially ranking images using comparison between each pair of scores and compute area under curve based on the ranking. Thus, the binary variables are $\mathbf{b} = (b_{1,2}, b_{1,3}, ...; b_{2,3}, b_{2,4}, ...; ...; b_{n-1,n})$, where $b_{i,j} = [c_{i,j} > 0] = [s_i - s_j > 0]$.

Some binary variables are however not contributing to the performance metric AP. For example, if y_i and y_j are both negatives, whether $b_{i,j}$ is 0 or 1 does not affect the AP and thus should be pruned off. This pruning can reduce the dimensionality of the variate. After pruning, our new set of binary variables are $\mathbf{b} = (b_{i,j})$, where i belongs to positives and j belongs to negatives.

When sampling the anchors, the best anchor derived from the ground truth is that all positive examples have higher scores than negative examples—all $b_{i,j} = 1$. As mentioned in Sec. 3.5, good anchors are generated by randomly flipping one binary bit from the best anchor. Bad anchors are sampled randomly. Nearby anchors are generated by flipping one bit from the current configuration towards the best anchor, which is adaptively depending on the current network parameters and training examples. We sample 16 anchors for each type in all of our experiments.

Table 1: PCKh of Stacked Hourglass with MSE and UniLoss on the validation set of MPII.

Loss	MSE $\sigma = 0.1$	$\sigma = 0.5$	$\sigma = 0.7$	$\sigma = 1$	$\sigma = 3$	$\sigma = 5$	$\sigma = 10$	UniLoss
PCKh	91.31	95.13	93.06	95.71	95.74	94.99	92.25	**95.77**

We use a 3-layer fully-connected neural network with 500 and 300 neurons in each hidden layer respectively. Our baseline model is trained with a 2-class cross-entropy loss. We train both baseline and UniLoss with a fixed learning rate of 0.01 for 30 epochs.

Results UniLoss (0.9988) performs similarly as the baseline cross-entropy loss (0.9989). This demonstrates that UniLoss can effectively optimize for a performance metric (AP) that is complicated to compute and involves a batch of images.

4.2 PCKh for Single-Person Pose Estimation

Task and Metric Single-person pose estimation predicts localization of human joints. We use MPII (Andriluka et al., 2014) which has around 22K image for training and 3K images for testing. For simplicity, we perform experiments on the joints of head only, but our method could be applied on an arbitrary number of human joints without any modification. Following prior work, we use PCKh (Percentage of Correct Keypoints wrt to head size) as the performance metric. It computes the percentage of the predicted joints that are within a given radius r of the ground truth. The radius is half of the head segment length.

Implementation Details Assume the network generates a mini-batch of heatmaps $\mathbf{s} = (\mathbf{s}^1, \mathbf{s}^2, \ldots, \mathbf{s}^n) \in \mathbf{R}^{n \times m}$, where n is the batch size, m is the number of pixels in each image. The pixel with the highest score in each heatmap is predicted as a key point during evaluation. We note the pixels within the radius r around the ground truth as positive pixels, and other pixels as negative. In PCKh, an outputed heatmap gives correct prediction if one of the positive pixels has the highest score in its heatmap. We thus need the comparisons of pixel-scores between all pairs of positive and negative pixels in each heatmap $\mathbf{s}^k$, where $\mathbf{s}^k = (s^k_{pos,1}, s^k_{pos,2}, \cdots, s^k_{pos,m_k}, s^k_{neg,1}, s^k_{neg,2}, \cdots, s^k_{neg,m-m_k})$ for all $1 \leq k \leq n$. m_k is the number of positive pixels in the k-th heatmap, and $s^k_{pos,j}$ ($s^k_{neg,j}$) is the score of the j-th positive (negative) pixel in the k-th heatmap. Then we have $\mathbf{b} = (b_{k,i,j})$ for $1 \leq k \leq n$, $1 \leq i \leq m_k$, $1 \leq j \leq m - m_k$, where $b_{k,i,j} = [s^k_{pos,i} - s^k_{neg,j} > 0]$, i.e. the comparison between the i-th positive pixel and the j-th negative pixel in the k-th heatmap.

The best anchor is with all binary values to be 1, meaning that the scores of all positive pixels are larger than the negative ones. For good anchors, we flip a small number of bits from the best. Bad anchors are randomly sampled. Nearby anchors are flipped from the current configuration by randomly picking a positive/negative pixel in the current output heatmaps and flipping all bits associated with this pixel. This generates nearby anchors that are slightly different from the current configuration.

We adopt the Stacked Hourglass Network (Newell et al., 2016) as our model. The baseline loss is the Mean Squared Error (MSE) between the predicted heatmaps and the manually-designed "ground truth" heatmaps. We train a single-stack hourglass network for both UniLoss and MSE using RMSProp (Hinton et al., 2012) with an initial learning rate 2.5e-4 for 30 epochs and then drop it by 4 for every 10 epochs until 50 epochs.

Results It is notable that the manual design of the target heatmaps is a part of MSE loss function for pose estimation. If we intuitively set the pixels at the exact joints to be 1 and the rest of pixels as 0 in the heatmaps, the training diverges. Luckily, Tompson et al. (2014) proposed to design target heatmaps as a 2D Gaussian bump centered on the ground truth joints, whose shape is controlled by its variance σ and the bump size. The success of the MSE loss function relies on the choices of σ and the bump size. UniLoss, on the other hand, requires no such design.

Table 2: Accuracy of ResNet-20 with CE loss and UniLoss on the CIFAR-10 and CIFAR-100 test set.

Loss	CIFAR-10	CIFAR-100
CE (cross-entropy) Loss	91.49	65.90
UniLoss	91.64	65.92

As shown in Table 1, our UniLoss achieves a 95.77 PCKh which is comparable as the 95.74 PCKh for MSE with the best σ. This validates the effectiveness of UniLoss in optimizing PCKh, a performance metric for structured network outputs.

We also observe that the baseline is sensitive to the shape of 2D Gaussian, as in Table 1. Smaller σ makes target heatmaps concentrated on ground truth joints and makes the optimization to be unstable. Larger σ generates vague training targets and decreases the performance. This demonstrates that conventional losses require dedicated manual design while UniLoss can be applied directly.

4.3 Accuracy for Multi-class Classification

Task and Metric Multi-class classification is a common task that has well-established conventional loss — cross-entropy loss. We use this task to show that in addition to above tasks, UniLoss also performs comparable in such common setting. Here we use CIFAR-10 and CIFAR-100 (Krizhevsky & Hinton, 2009), with 32×32 images and 10/100 classes. They each has 50k training images and 10k test images. Following prior work (He et al., 2016), we split the training set into a 45k-5k train-validation split. We use accuracy (the percentage of correctly classified images) as our metric.

Implementation Details As shown in Sec. 3.2, given the output of a mini-batch of n images $\mathrm{s} = (s_{1,1}, s_{1,2}.., s_{n,p})$, we compare the score of the ground truth class and the scores of other $p-1$ classes for each image. That is, for the i-th image with the ground truth label y_i, $b_{i,j} = [s_{i,y_i} - s_{i,j} > 0]$, where $1 \leq j \leq p$, $j \neq y_i$, and $1 \leq i \leq n$. In our experiments, batch size n is 128 and p are 10/100.

The best anchor derived from the ground truth is with all $b_{i,j} = 1$, meaning that for each image, the ground truth class has higher score than every other class. Good anchors are generated by randomly flipping one binary bit from the best anchor. Bad anchors are sampled randomly. Nearby anchors are generated by flipping one bit from the current configuration towards the best anchor. For tasks with many binary variables such as CIFAR-100, we train 20% of all binary variables in each update to accelerate training.

We use the ResNet-20 architecture (He et al., 2016). Our baselines are trained with cross-entropy (CE) loss. All experiments are trained following the same augmentation and pre-processing techniques as in prior work (He et al., 2016). We use an initial learning rate of 0.1, divided by 10 and 100 at the 140 epoch and the 160 epoch, with a total of 200 epochs trained for both baseline and UniLoss on CIFAR-10. On CIFAR-100, we train baseline with the same training schedule and UniLoss with 5x training schedule because we only train 20% binary variables at each step. For fair comparison, we also train baseline with the 5x training schedule but observes no further improvement.

Results Our implementation of the baseline method obtains a slightly better accuracy (91.49%) than that was reported in He et al. (2016)—91.25% on CIFAR-10 and obtains 65.9% on CIFAR-100. UniLoss performs similarly (91.64% and 65.92%) as baseline on both datasets(Table 2), which shows that even when the conventional loss is well-established for the particular task and metric, UniLoss still matches the conventional loss.

5 Conclusion

We have introduced UniLoss, a method for generating surrogate losses in a unified way, reducing the amount of manual design of task-specific surrogate losses. The proposed framework is based on the observation that there exists a common re-factorization of the evaluation computation for many tasks and performance metrics. We demonstrate that using UniLoss, we can optimize for various tasks and performance metrics, achieving comparable performance as task-specific losses.

REFERENCES

Erin L Allwein, Robert E Schapire, and Yoram Singer. Reducing multiclass to binary: A unifying approach for margin classifiers. In *ICML*, 2000.

Mykhaylo Andriluka, Leonid Pishchulin, Peter Gehler, and Bernt Schiele. 2d human pose estimation: New benchmark and state of the art analysis. In *CVPR*, 2014.

Peter L Bartlett, Michael I Jordan, and Jon D McAuliffe. Convexity, classification, and risk bounds. *Journal of the American Statistical Association*, 101(473):138–156, 2006.

Juan C Caicedo and Svetlana Lazebnik. Active object localization with deep reinforcement learning. In *ICCV*, 2015.

Koby Crammer and Yoram Singer. On the algorithmic implementation of multiclass kernel-based vector machines. In *ICML*, 2001.

Tamir Hazan, Joseph Keshet, and David A McAllester. Direct loss minimization for structured prediction. In *NeurIPS*, 2010.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016.

Paul Henderson and Vittorio Ferrari. End-to-end training of object class detectors for mean average precision. In *ACCV*, 2016.

Geoffrey Hinton, Nitish Srivastava, and Kevin Swersky. Lecture 6a overview of mini–batch gradient descent. *Coursera Lecture slides https://class. coursera. org/neuralnets-2012-001/lecture,[Online*, 2012.

Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. 2009.

Siqi Liu, Zhenhai Zhu, Ning Ye, Sergio Guadarrama, and Kevin Murphy. Improved image captioning via policy gradient optimization of spider. In *ICCV*, 2017.

Weiyang Liu, Yandong Wen, Zhiding Yu, and Meng Yang. Large-margin softmax loss for convolutional neural networks. In *ICML*, 2016.

Alejandro Newell and Jia Deng. Pixels to graphs by associative embedding. In *NeurIPS*, 2017.

Alejandro Newell, Kaiyu Yang, and Jia Deng. Stacked hourglass networks for human pose estimation. In *ECCV*, 2016.

Alejandro Newell, Zhiao Huang, and Jia Deng. Associative embedding: End-to-end learning for joint detection and grouping. In *NeurIPS*, 2017.

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

Harish G Ramaswamy, Shivani Agarwal, and Ambuj Tewari. Convex calibrated surrogates for low-rank loss matrices with applications to subset ranking losses. In *NeurIPS*, 2013.

Harish G Ramaswamy, Balaji Srinivasan Babu, Shivani Agarwal, and Robert C Williamson. On the consistency of output code based learning algorithms for multiclass learning problems. In *Conference on Learning Theory*, pp. 885–902, 2014.

Marc'Aurelio Ranzato, Sumit Chopra, Michael Auli, and Wojciech Zaremba. Sequence level training with recurrent neural networks. In *ICLR*, 2016.

Joseph Redmon and Ali Farhadi. Yolo9000: Better, faster, stronger. In *CVPR*, 2017.

Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *CVPR*, 2016.

Donald Shepard. A two-dimensional interpolation function for irregularly-spaced data. In *Proceedings of the 23rd ACM National Conference*. ACM, 1968.

Yang Song, Alexander Schwing, Raquel Urtasun, et al. Training deep neural networks via direct loss minimization. In *ICML*, 2016.

Richard S Sutton, David A McAllester, Satinder P Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *NeurIPS*, 2000.

Michael Taylor, John Guiver, Stephen Robertson, and Tom Minka. Softrank: optimizing non-smooth rank metrics. In *WSDM*, 2008.

Ambuj Tewari and Peter L Bartlett. On the consistency of multiclass classification methods. In *ICML*, 2007.

Jonathan J Tompson, Arjun Jain, Yann LeCun, and Christoph Bregler. Joint training of a convolutional network and a graphical model for human pose estimation. In *NeurIPS*, 2014.

Ioannis Tsochantaridis, Thorsten Joachims, Thomas Hofmann, and Yasemin Altun. Large margin methods for structured and interdependent output variables. In *ICML*, 2005.

Serena Yeung, Olga Russakovsky, Greg Mori, and Li Fei-Fei. End-to-end learning of action detection from frame glimpses in videos. In *CVPR*, 2016.

Tong Zhang. Statistical behavior and consistency of classification methods based on convex risk minimization. *Annals of Statistics*, 2004.