Area Under the ROC Curve Maximization for Metric Learning

Bojana Gajić
Vintra, Inc.
Barcelona, Spain
bgajic@vintra.io

Ariel Amato
Vintra, Inc.
Barcelona, Spain
aamato@vintra.io

Ramon Baldrich
Computer Vision Center
Barcelona, Spain
aamato@vintra.io

Joost van de Weijer
Computer Vision Center
Barcelona, Spain
aamato@vintra.io

Carlo Gatta
Vintra, Inc.
Barcelona, Spain
carlo.gatta@gmail.com

1. AUC implementation details

The AUC loss function has three input parameters: (1) list of \(N\) features extracted from the backbone architecture, (2) a list of ground truth labels that correspond to the features, and (3) the step size \(\Delta s\) (line 1 in algorithm 1). We first calculate the similarities between all features from the minibatch (line 2 in algorithm 1). The similarity matrix is a square matrix of size \(N \times N\), where the value of similarity \([i, j]\) corresponds to the similarity between features \(i\) and \(j\).

We get the positive mask as a binary matrix of size \(N \times N\) that indicates if the elements in the similarity matrix belong to different images from the same class. Similarly, the negative mask extracts the positions in the similarity matrix that belong to different classes. We mask the similarity matrix by element-wise multiplying it with positive mask and find the minimal similarity per each row (line 3 in algorithm 1). In this way, we obtain a vector of the hardest positive similarities (HPS) for each input feature. Similarly, we mask the similarity matrix with the negative mask and find the maximum similarity per row, resulting in a vector of size \(N\) of hardest negative similarities (HNS) for each input feature (line 4 in algorithm 1).

We define a vector of thresholds as a step vector of size \(S + 1: [t_{\min}, t_{\min} + \Delta s, ..., t_{\max}]\) (line 5 in algorithm 1), and get the optimal slope for the given \(\Delta s\) from Table 2 (line 6 in algorithm 1). For each threshold from the step vector and for each hardest positive similarity, we get a value of sigmoid defined in formula 6, and store it in a matrix \(\sigma^+_r\) of size \(N \times (S + 1)\) (line 7 in algorithm 1). Similarly, we obtain the \(\sigma^-_r\) matrix based on hardest negative similarities (line 8 in algorithm 1).

We obtain vectors \(s_1\) and \(s_2\) of length \(S\) from \(\sigma^+_r\) and \(\sigma^-_r\) matrices (lines 9-13 in algorithm 1). Although we present the algorithm with a for loop over the samples, this procedure is implemented with matrices and in a parallel way exploiting the GPU parallelization capabilities. We get the estimated area under the ROC curve \(R\) based on the samples from mini-batch, as shown in line 14 of algorithm 1. Finally, the AUC loss is calculated as \(1 - R\) (line 15 in algorithm 1).

2. Limitations

The main limitation of the AUC loss is its performance on small datasets. In Table 1 we show that AUC performs well on small datasets as well. We trained our model in two scenarios: using original images and using crops of the birds provided by the authors of the dataset. We treated both cases in the same way, resizing the images to 256x256 and using a random 224x224 crop for training, and central for testing. AUC loss achieves results that are comparable with state of the art when trained with original images. Our method outperforms all ensemble-based methods, and shows comparable results with the newest state-of-the-art methods. The only method that performs significantly better is RankMI [4]. Even though RankMI outperforms AUC, it is computationally more expensive, as the model is built out of two networks that are updated alternately, it has two extra hyper parameters; also the authors do not report the input image size. R-Margin model achieves 6.7% higher rank@1 on CUB-200 dataset, while using a bigger mini-batch, distance based tuple mining, and \(\rho\) regularization. Additionally, this model has an extra hyper parameter \(\beta\) and the results vary significantly with different initialization values. We believe that AUC loss leads to overfitting due to its strong gradients, when trained on a small size datasets. We improved the performance of AUC by using image crops instead of whole images (see Table 2).
Algorithm 1 $AUC_{BH}$ loss function

1: **Input**: $features, labels, \Delta s$
2: calculate the matrix of similarities based on features
3: find the lowest similarity per row for features that belong to the same class (hardest_positive_similarity or $HPS$)
4: find the highest similarity per row for features that do not belong to the same class (hardest_negative_similarity or $HNS$)
5: $step\_vector \leftarrow \{t_{\text{min}}, t_{\text{min}} + \Delta s, ..., t_{\text{max}} - \Delta s\}$
6: for a given $\Delta s$, find optimal slope from Table 6
7: $\sigma^+ \leftarrow \frac{1}{1 + e^{-\text{slope}(HPS - \text{step\_vector})}}$
8: $\sigma^- \leftarrow \frac{1}{1 + e^{-\text{slope}(HNS - \text{step\_vector})}}$
9: for $j$ in $1, 2, ..., S$
10: $s \leftarrow \text{step\_vector}[j]$
11: $s_1[j] \leftarrow \sum_{i=1}^{N} \sigma^+[i, s] + \sigma^-+[i, s + \Delta s]$
12: $s_2[j] \leftarrow \sum_{i=1}^{N} \sigma^--[i, s] - \sigma^+[i, s + \Delta s]$
13: end for
14: $R \leftarrow \frac{1}{2N^2} \sum_{j=1}^{S} s_1[j]s_2[j]$
15: $AUC \leftarrow 1 - R$

References


