Loss Decomposition for Fast Learning in Large Output Spaces

Ian E.H. Yen [1] **Satyen Kale** [2] **Felix X. Yu** [2] **Dan Holtmann-Rice** [2] **Sanjiv Kumar** [2] **Pradeep Ravikumar** [1]

Abstract

For problems with large output spaces, evaluation of the loss function and its gradient are expensive, typically taking linear time in the size of the output space. Recently, methods have been developed to speed up learning via efficient data structures for *Nearest-Neighbor Search (NNS)* or *Maximum Inner-Product Search (MIPS)*. However, the performance of such data structures typically degrades in high dimensions. In this work, we propose a novel technique to reduce the intractable high dimensional search problem to several much more tractable lower dimensional ones via *dual decomposition* of the loss function. At the same time, we demonstrate guaranteed convergence to the original loss via a greedy message passing procedure. In our experiments on multiclass and multilabel classification with hundreds of thousands of classes, as well as training skip-gram word embeddings with a vocabulary size of half a million, our technique consistently improves the accuracy of search-based gradient approximation methods and outperforms sampling-based gradient approximation methods by a large margin.

1. Introduction

Large output spaces are ubiquitous in several machine learning problems today: for example, extreme multiclass or multilabel classification problems with many classes, language modeling with big vocabularies, or metric learning with a large number of pairwise distance constraints. In all such problems, a key bottleneck in training models is evaluation of the loss function and its gradient. The loss functions used for such problems typically require an enumeration of all the possible outputs, and thus, naïvely, necessitate a linear running time in the number of outputs for evaluation. This can be a significant bottleneck in iterative methods such as gradient descent used to train the model, since each step now requires a huge number of operations.

Many approaches have been proposed to mitigate this issue. One body of work imposes structure over the output space, such as low-rank (Yu et al., 2014), tree-structure (Prabhu & Varma, 2014), locally low-rank (Bhatia et al., 2015), or hierarchical factorization (Morin & Bengio, 2005; Mnih & Hinton, 2009). However, structural assumptions can be violated in many situations. For example, while the low-rank structure is typically reasonable in a recommendation problem, it is usually not true in multiclass classification as for each instance there is exactly one correct answer (i.e. classes may not be correlated with each other). Additionally, even for valid structural assumptions, constructing the correct structure from data is hard, and in practice heuristics or human annotation are required (Morin & Bengio, 2005; Mnih & Hinton, 2009).

Another approach is sampling approximation (Mikolov et al., 2013; Gutmann & Hyvärinen, 2012; Jean et al., 2014), which computes an estimate of the gradient based on the scores of only a small fraction of the negative output classes and also a small set of classes labeled as positive. The approximation, however, has large variance when the loss has a skewed distribution over classes. For example, in extreme multiclass or multilabel classification, the loss typically only concentrates on a few confusing classes, which have small probabilities of being sampled. The variance in gradient estimation often leads to slow progress of the learning algorithm.

In this paper, we consider problems with large output spaces, but with each example having only a relatively small set of correct outputs. The learning objective for such tasks typically has its gradient concentrated on a relatively small number of classes, and therefore an efficient way to learn is to search for classes of significant gradient magnitude. For example, (Yen et al., 2016; 2017) proposed a method to search classes efficiently by maintaining a sparse model during training. However, this method applies only in problems of high input dimension. Another strategy that has received a lot of attention recently is to utilize data structures to find classes efficiently through *Maximum Inner-Product Search (MIPS)* or *Nearest-Neighbor*

[1]Carnegie Mellon University, Pittsburgh, USA [2]Google, New York, USA. Correspondence to: Ian E.H. Yen <eyan@cs.cmu.edu>, Satyen Kale <satyenkale@google.com>.

Proceedings of the 34^{th} International Conference on Machine Learning, Sydney, Australia, PMLR 70, 2017.

Search (NNS) (Yen et al., 2013; Vijayanarasimhan et al., 2014; Mussmann & Ermon, 2016; Mussmann et al., 2017; Spring & Shrivastava, 2017b;a; Wu et al., 2017; Guo et al., 2016). The main challenge here is that as dimension grows, it becomes difficult to perform MIPS or NNS with both high recall and high precision, and therefore gradient approximation through MIPS or NNS often sacrifices accuracy to achieve efficiency.

In this work, we propose an algorithm based on an application of *dual decomposition* (Boyd et al., 2011) to the convex-conjugate representation of the loss function. This can be viewed as a complementary technique for applying search data structures to a learning problem. Essentially, the algorithm replaces the high dimensional search problem with several lower dimensional searches by decoupling the dimensions via dual decomposition. Lower dimensional search can be done much more efficiently, and the different searches are then coupled together via a greedy message passing algorithm. We prove that this greedy message passing technique is guaranteed to converge and thus we can obtain good approximations to the loss and its gradient. We term our overall approach LDGS for Loss Decomposition Guided Search.

Our experiments on large-scale face recognition, document tagging and word embedding show that the proposed approach significantly improves the accuracy of the search-based gradient approximation method and is orders of magnitude faster than other strategies of gradient approximation such as sampling.

2. Problem Setup

Let $\mathcal{X}$ denote the input space and $\mathcal{Y}$ the output space, and let $K := |\mathcal{Y}|$. In this paper we focus on the situation where K is extremely large, on the order of hundreds of thousands or larger. We are interested in learning a scoring function $f : \mathcal{X} \to \mathbb{R}^K$ for a large output space $\mathcal{Y}$ from a given class of such functions, $\mathcal{F}$. Labeled samples are pairs $(\boldsymbol{x}, \mathcal{P})$ with $\boldsymbol{x} \in \mathcal{X}$ and $\mathcal{P} \subseteq \mathcal{Y}$ which denotes the set of correct labels for the input point $\boldsymbol{x}$. We use the notation $\mathcal{N} := \mathcal{Y} \setminus \mathcal{P}$ to denote the set of negative labels for the example. Given a collection of training samples $\{(\boldsymbol{x}_i, \mathcal{P}_i)\}_{i=1}^N$, the learning objective takes the following form:

$$\min_{f \in \mathcal{F}} \frac{1}{N} \sum_{i=1}^{N} L(f(\boldsymbol{x}_i), \mathcal{P}_i).$$

where $L : \mathbb{R}^K \times 2^{\mathcal{Y}} \to \mathbb{R}$ is a loss function such that $L(\boldsymbol{z}, \mathcal{P})$ penalizes the discrepancy between the score vector $\boldsymbol{z} \in \mathbb{R}^K$ and a set of positive labels $\mathcal{P} \subseteq \mathcal{Y}$. The evaluation of the loss function and its gradient with respect to the score vector, $\nabla_{\boldsymbol{z}} L(\boldsymbol{z}, \mathcal{P})$, typically has cost growing linearly with the size of the output space K, and thus is expensive for problems with huge output spaces.

The key to our method for reducing the complexity of loss and gradient evaluation will be the following linear structural assumption on the class of scoring functions $\mathcal{F}$: there is an *embedding dimension* parameter $D \in \mathbb{N}$ such that for every $f \in \mathcal{F}$, we can associate a weight matrix $\boldsymbol{W} \in \mathbb{R}^{K \times D}$ and feature map $\boldsymbol{\phi} : \mathcal{X} \to \mathbb{R}^D$ so that for all $\boldsymbol{x} \in \mathcal{X}$,

$$f(\boldsymbol{x}) = \boldsymbol{W}\boldsymbol{\phi}(\boldsymbol{x}). \tag{1}$$

We will assume that $D \ll K$, say on the order of a few hundreds or thousands, so that we can explicitly evaluate $\boldsymbol{\phi}(\boldsymbol{x})$.

The problem we consider is the following: given f and a batch of samples $\{\boldsymbol{x}_i, \mathcal{P}_i\}_{i=1}^N$, compute an approximation to the empirical loss $\frac{1}{N}\sum_{i=1}^N L(f(\boldsymbol{x}_i), \mathcal{P}_i)$ and its gradient. This is an important subroutine that naturally arises in either full batch gradient descent or minibatch stochastic gradient descent.

The main challenge here is to construct data structures that preprocess the matrix $\boldsymbol{W}$ so that good approximations to the loss $f(\boldsymbol{x}_i, \mathcal{P}_i)$ and its gradient can be computed without computing the vector $f(\boldsymbol{x})$ entirely: i.e. we desire sublinear (in K) time computation of such approximations given access to an appropriate data structure.

Before proceeding to our dual decomposition based search technique, we give a few examples of problems with large output space that fit in our framework:

1. **Extreme Classification.** In extreme classification problems, popular classification loss functions include *Cross-Entropy Loss*

$$L(\boldsymbol{z}, \mathcal{P}) := \sum_{k \in \mathcal{P}} \log\left(\textstyle\sum_{j=1}^K \exp(z_j)\right) - z_k \tag{2}$$

and *Max-Margin Loss*

$$L(\boldsymbol{z}, \mathcal{P}) := \left[\max_{k \in \mathcal{P}, j \in \mathcal{N}} z_k - z_j + 1\right]_+. \tag{3}$$

For multiclass problems, $|\mathcal{P}| = 1$, while for multilabel problems we usually have $|\mathcal{P}| \ll K$. A typical scoring function takes the form

$$f(\boldsymbol{x}) := \boldsymbol{W}\boldsymbol{\phi}(\boldsymbol{x}). \tag{4}$$

Here, $\boldsymbol{\phi}(\boldsymbol{x})$ is a feature map constructed either from the domain knowledge or via learning (e.g., a neural network). Both of them fit the structural assumption (1).

2. **Metric Learning.** In Metric Learning problems, during training we learn a function

$$f(\boldsymbol{x}) = [-d(\boldsymbol{x}, \boldsymbol{y})]_{\boldsymbol{y} \in \mathcal{Y}}, \tag{5}$$

that denotes the dissimilarities of the point $\boldsymbol{x}$ to a collection of points $\boldsymbol{y} \in \mathcal{Y}$. Common choices of the dissimilarity function include the squared Euclidean distance $d(\boldsymbol{x}, \boldsymbol{y}) = \|\boldsymbol{\psi}(\boldsymbol{x}) - \boldsymbol{\psi}(\boldsymbol{y})\|_2^2$ parameterized by a nonlinear transformation $\boldsymbol{\psi} : \mathcal{X} \to \mathbb{R}^d$ for some $d \in \mathbb{N}$, and, more generally, the squared Mahalanobis distance $d(\boldsymbol{x}, \boldsymbol{y}) = (\boldsymbol{\psi}(\boldsymbol{x}) - \boldsymbol{\psi}(\boldsymbol{y}))^\top \boldsymbol{M}(\boldsymbol{\psi}(\boldsymbol{x}) - \boldsymbol{\psi}(\boldsymbol{y}))$ parameterized by $\boldsymbol{\psi}$ and a positive definite matrix $\boldsymbol{M}$. The candidate set $\mathcal{Y}$ could be the whole set of training samples $\{\boldsymbol{x}_i\}_{i=1}^N$, or a collection of latent proxies $\{\boldsymbol{y}_k\}_{k=1}^K$ as suggested by a recent state-of-the-art method (Movshovitz-Attias et al., 2017). For each sample $(\boldsymbol{x}, \mathcal{P})$, the goal is to learn a distance function s.t. the positive candidates $\mathcal{P}$ are closer to $\boldsymbol{x}$ than the negative ones. Common loss functions for the task are *Neighborhood Component Analysis (NCA) loss* (Goldberger et al., 2005)

$$L(\boldsymbol{z}, \mathcal{P}) := \sum_{k \in \mathcal{P}} \log\left(\textstyle\sum_{j=1}^K \exp(z_j)\right) - z_k \qquad (6)$$

and the *Triplet loss* (Weinberger & Saul, 2009)

$$L(\boldsymbol{z}, \mathcal{P}) = \sum_{k \in \mathcal{P}} \sum_{j \in \mathcal{N}} [z_k - z_j + 1]_+. \qquad (7)$$

It is easy to see that such scoring functions satisfy the structural assumption (1): for the scoring function f given by the squared Mahalanobis distance parameterized by $\boldsymbol{\psi}$ and $\boldsymbol{M}$, the matrix $\boldsymbol{W}$ consists of the rows $\langle -\boldsymbol{\psi}(\boldsymbol{y})^\top \boldsymbol{M} \boldsymbol{\psi}(\boldsymbol{y}), 2\boldsymbol{\psi}(\boldsymbol{y})^\top \boldsymbol{M}, -1\rangle$ for each $\boldsymbol{y} \in \mathcal{Y}$, and $\boldsymbol{\phi}(\boldsymbol{x}) = \langle 1, \boldsymbol{\psi}(\boldsymbol{x})^\top, \boldsymbol{\psi}(\boldsymbol{x})^\top \boldsymbol{M} \boldsymbol{\psi}(\boldsymbol{x})\rangle^\top$. Thus the embedding dimension $D = d + 2$.

3. **Word Embeddings.** In the standard *word2vec* training (Mikolov et al., 2013), the input space $\mathcal{X}$ is the vocabulary set, and the output space $\mathcal{Y} = \mathcal{X}$; thus K is the vocabulary size. The *Skip-gram* objective learns a scoring function f of the following form:

$$f(\boldsymbol{x}) = \langle \boldsymbol{\phi}(\boldsymbol{y})^\top \boldsymbol{\phi}(\boldsymbol{x}) \rangle_{\boldsymbol{y} \in \mathcal{X}}, \qquad (8)$$

where $\boldsymbol{\phi}(\cdot)$ is a latent word embedding. This clearly fits the structural assumption (1): the rows of the matrix $\boldsymbol{W}$ are the embeddings $\boldsymbol{\phi}(\boldsymbol{y})$ for all $\boldsymbol{y} \in \mathcal{X}$.

Then given a text corpus $\mathcal{D}$, the loss function[1] for a sample $(\boldsymbol{x}, \mathcal{P})$ where $\mathcal{P}$ is the set of words in the corpus appearing within a certain size window around the input word $\boldsymbol{x}$, is given by

$$L(\boldsymbol{z}, \mathcal{P}) = q_{\boldsymbol{x}} \sum_{\boldsymbol{y} \in \mathcal{P}} q_{\boldsymbol{y}|\boldsymbol{x}} \cdot [\log\left(\textstyle\sum_{\boldsymbol{y}' \in \mathcal{X}} \exp(z_{\boldsymbol{y}'})\right) - z_{\boldsymbol{y}}] \qquad (9)$$

where $q_{\boldsymbol{x}}$ is the empirical unigram frequency of $\boldsymbol{x}$ and $q_{\boldsymbol{y}|\boldsymbol{x}}$ is the empirical frequency of observing $\boldsymbol{y}$ within a window of $\boldsymbol{x}$ in the corpus $\mathcal{D}$.

[1] This is a more compact reformulation of the loss function in (Mikolov et al., 2013).

Algorithm 1 Loss and Gradient Approximation via Search

input A sample $(\boldsymbol{x}, \mathcal{P})$, accuracy parameter $\tau > 0$, and access to a MIPS data structure $\mathcal{T}$ for the rows of $\boldsymbol{W}$.
output Approximations to $L(f(\boldsymbol{x}), \mathcal{P})$, $\nabla L(f(\boldsymbol{x}), \mathcal{P})$.

1: Query $\mathcal{T}$ with $\phi(\boldsymbol{x})$ and threshold τ to find $\mathcal{S} := \{k \mid |[f(\boldsymbol{x})]_k| > \tau\}$.
2: Construct a sparse approximation $\tilde{\boldsymbol{z}}$ for $f(\boldsymbol{x})$ by setting $\tilde{z}_k = f(\boldsymbol{x})_k$ for $k \in \mathcal{S} \cup \mathcal{P}$, and $\tilde{z}_k = 0$ for $k \notin \mathcal{S} \cup \mathcal{P}$.
3: Return $L(\tilde{\boldsymbol{z}}, \mathcal{P})$ and $\nabla L(\tilde{\boldsymbol{z}}, \mathcal{P})$.

2.1. Loss and Gradient Approximation via Search

All the loss functions we considered in the applications mentioned share a key feature: their value can be well approximated by the scores of the positive labels and the *largest* scores of the negative labels. Similarly, their gradients are dominated by the coordinates corresponding to the positive labels and the negative labels with the largest scores. For example, the *Max-Margin loss* (3) is completely determined by the largest score of the negative labels and the lowest scores of the positive labels, and its gradient is non-zero only on the negative label with largest score and the positive label with lowest score. Similarly, for the *Cross-Entropy loss* (2), the coordinates of the gradient corresponding to the negative classes are dominated by the ones with the highest score; the gradient coordinates decrease exponentially as the scores decrease.

This key property suggests the following natural idea for approximating these losses and their gradients: since the score function f satisfies the linear structural property (1), we can compute the largest scores efficiently via a *Maximum Inner Product Search* (MIPS) data structure (Shrivastava & Li, 2014). This data structure stores a large data set of vectors $\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_K \in \mathbb{R}^D$ and supports queries of the following form: given a target vector $\boldsymbol{u} \in \mathbb{R}^D$ and a threshold τ, it returns the vectors $\boldsymbol{v}_i$ stored in it that satisfy $|\boldsymbol{v}_i^\top \boldsymbol{u}| \geq \tau$ in time that is typically sublinear in K. Thus, we can preprocess $\boldsymbol{W}$ by storing the rows of $\boldsymbol{W}$ in an efficient MIPS data structure. Then for each sample $\boldsymbol{x}$, we can compute the highest scores by querying this data structure with the target vector $\boldsymbol{\phi}(\boldsymbol{x})$ and some reasonable threshold τ, computing approximations to the loss and gradient from the returned vectors (and treating all other scores as 0). This method is depicted in Algorithm 1.

The error in this approximation is naturally bounded by τ times the ℓ_∞ Lipschitz constant of $L(\cdot, \mathcal{P})$. For most loss functions considered in this paper, the ℓ_∞ Lipschitz constant is reasonably small: 2 for *Max-Margin loss*, $O(P_{\max} \log(K))$ for *Cross-Entropy loss* (here, $P_{\max}$ is the maximum number of positive labels for any example), etc.

The main difficulty in applying this approach in practice

is the curse of dimensionality: the dependence on D is *exponential* for exact methods, and even for approximate methods, such as Locality-Sensitive Hashing, the cost still implicitly depends on the dimension as points become far apart when the intrinsic dimensionality is high (Li & Malik, 2017).

To deal with the curse of dimensionality, we introduce a novel search technique based on dual decomposition. This method, and its analysis, are given in the following section.

In order to apply and analyze the technique, we need the loss functions to be smooth (i.e. have Lipschitz continuous gradients). For non-smooth losses like *Max-Margin loss* (3), we apply Nesterov's smoothing technique (Nesterov, 2005), which constructs a surrogate loss function with guaranteed approximation quality by adding a strongly convex term to the Fenchel conjugate of the loss:

$$L_\mu(\boldsymbol{z}) := \max_{\boldsymbol{\alpha}} \langle \boldsymbol{z}, \boldsymbol{\alpha} \rangle - \left(L^*(\boldsymbol{\alpha}) + \frac{\mu}{2} \|\boldsymbol{\alpha}\|^2 \right). \quad (10)$$

Here, μ is a smoothing parameter that ensures that the surrogate loss has $\frac{1}{\mu}$ Lipschitz continuous gradients while approximating the original loss function to within $O(\mu)$. This *Smoothed Max-Margin loss* has gradient

$$\nabla L(\boldsymbol{z}) := \mathbf{proj}_{\mathcal{C}}(\tfrac{\boldsymbol{z}+\mathbf{1}_{\mathcal{N}}}{\mu}) \quad (11)$$

where $\mathbf{1}_{\mathcal{N}}$ denotes a vector containing 0 for indices $k \in \mathcal{P}$ and 1 for $k \in \mathcal{N}$, and $\mathbf{proj}_{\mathcal{C}}(.)$ denotes the projection onto the bi-simplex $\mathcal{C} = \{\boldsymbol{\alpha} \mid \sum_{k\in\mathcal{N}} \alpha_k = \sum_{k\in\mathcal{P}} -\alpha_k \leq 1,\ \alpha_{\mathcal{N}} \geq 0, \alpha_{\mathcal{P}} \leq 0\}$. The Smoothed Max-Margin loss and its gradient can again be computed using the largest few scores.

3. Loss Decomposition

We now describe our loss decomposition method. Recall the linear structural assumption (1): $f(\boldsymbol{x}) = \boldsymbol{W}\boldsymbol{\phi}(\boldsymbol{x})$ for all $\boldsymbol{x} \in \mathcal{X}$. In this section, we will keep $(\boldsymbol{x}, \mathcal{P})$ fixed, and we will drop the dependence on $\mathcal{P}$ in L for convenience and simply use the notation $L(f(\boldsymbol{x}))$ and $\nabla L(f(\boldsymbol{x}))$.

While MIPS over the D-dimensional rows of $\boldsymbol{W}$ can be computationally expensive, we can exploit the linear structure of f by decomposing it: chunking the D coordinates of the vectors in $\mathbb{R}^D$ into B blocks, each of size D/B. Here $B \in \mathbb{N}$ is an integer; larger B leads to easier MIPS problems but reduces accuracy of approximations produced. Let $\boldsymbol{W}^{(1)}, \boldsymbol{W}^{(2)}, \ldots, \boldsymbol{W}^{(B)}$ be the corresponding block partitioning of $\boldsymbol{W}$ obtained by grouping together the columns corresponding to the coordinates in each block. Similarly, let $\boldsymbol{\phi}^{(1)}(\boldsymbol{x}), \boldsymbol{\phi}^{(2)}(\boldsymbol{x}), \ldots, \boldsymbol{\phi}^{(B)}(\boldsymbol{x})$ be the conformal partitioning of the coordinates of $\boldsymbol{\phi}(\boldsymbol{x})$.

Now define the overall score vector $\boldsymbol{z} := f(\boldsymbol{x}) = \boldsymbol{W}\boldsymbol{\phi}(\boldsymbol{x})$, and per-chunk score vectors $\boldsymbol{z}_j = \boldsymbol{W}^{(j)}\boldsymbol{\phi}^{(i)}(\boldsymbol{x})$, for $j \in [B]$. Then we have $\boldsymbol{z} = \sum_{j=1}^{B} \boldsymbol{z}_j$, in other words, we have a decomposition of the score vector. The following theorem states that the loss of a decomposable score vector can itself be decomposed into several parts connected through a set of message variables. This theorem is key to decoupling the variables into lower dimensional chunks that can be optimized separately via an efficient MIPS data structure. While this theorem can be derived by applying dual decomposition to the convex conjugate of the loss function, here we provide a simpler direct proof by construction.

Theorem 1. *Let $L : \mathbb{R}^D \to \mathbb{R}$ be a convex function, and let $\boldsymbol{z} \in \mathbb{R}^D$ be decomposed as a sum of B vectors as follows: $\boldsymbol{z} = \sum_{j=1}^{B} \boldsymbol{z}_j$. Then $L(\boldsymbol{z})$ is equal to the optimum value of the following convex minimization problem:*

$$\min_{\boldsymbol{\lambda}_j \in \mathbb{R}^D,\ j\in[B]} \ \frac{1}{B}\sum_{j=1}^{B} L(B(\boldsymbol{z}_j + \boldsymbol{\lambda}_j))\ s.t.\ \sum_{j=1}^{B} \boldsymbol{\lambda}_j = 0. \quad (12)$$

Proof. First, for any $\boldsymbol{\lambda}_1, \boldsymbol{\lambda}_2, \ldots, \boldsymbol{\lambda}_B \in \mathbb{R}^D$ such that $\sum_{j=1}^{B} \boldsymbol{\lambda}_j = 0$, by Jensen's inequality applied to the convex function L, we have $L(\boldsymbol{z}) \leq \frac{1}{B}\sum_{j=1}^{B} L(B(\boldsymbol{z}_j + \boldsymbol{\lambda}_j))$. On the other hand, if we set $\boldsymbol{\lambda}_j = \frac{1}{B}\boldsymbol{z} - \boldsymbol{z}_j$ for all $j \in [B]$, we have $L(\boldsymbol{z}) = \frac{1}{B}\sum_{j=1}^{B} L(B(\boldsymbol{z}_j + \boldsymbol{\lambda}_j))$. □

3.1. Loss Decomposition Guided Search (LDGS)

Theorem (1) is the basis for our algorithm for computing approximations to the loss and its gradient. This approximation is computed by approximately solving the convex minimization problem (12) *without computing the whole score vector $\boldsymbol{z}$*, using a form of descent method on the $\boldsymbol{\lambda}_j$ variables (which we refer to as "message passing"). The gradient computations required for each step can be (approximately) done using an efficient MIPS data structure storing the D/B dimensional rows of $\boldsymbol{W}^{(j)}$. The details of the algorithm are given in Algorithm 2. It can be viewed as running a version of the Frank-Wolfe algorithm on an appropriate convex function.

A sublinear in K time implementation of step 5 in the algorithm relies on the fact that both $\tilde{\boldsymbol{z}}_j$ and $\boldsymbol{\lambda}_j$ are sparse vectors, which in turn relies on the fact that gradients of the loss functions of interest are either sparse or concentrated on a few coordinates. Step 9 in the algorithm moves the current solution towards the optimal solution $\boldsymbol{\lambda}_j^*$ that we have a closed form formula for, thanks to the constructive proof of Theorem (1). This movement is only done for the set of coordinates of the gradients of high magnitude identified in step 5 of the algorithm, thus ensuring that only a few coordinates are updated. Thus essentially the algorithm is performing a greedy descent towards the optimal solution. For more details on how the data structures are maintained in the algorithm, refer to Section 4.

Algorithm 2 Greedy Message Passing

input a sample $\boldsymbol{x}$, threshold parameters $\tau_1, \tau_2 > 0$, and access to B MIPS data structures $\mathcal{T}_j$ storing the rows of $\boldsymbol{W}^{(j)}$, for $j \in [B]$

output Approximation to $\nabla L(f(\boldsymbol{x}))$.

1: Query $\mathcal{T}_j$ with $\boldsymbol{\phi}^{(j)}(\boldsymbol{x})$ and threshold τ to find $\mathcal{S}_j := \{k \mid |[\boldsymbol{z}_j]_k| > \tau_1\}$.
2: Construct a sparse approximation $\tilde{\boldsymbol{z}}_j$ for $\boldsymbol{z}_j$ by setting $[\tilde{\boldsymbol{z}}_j]_k = [\boldsymbol{z}_j]_k$ for $k \in \mathcal{S}_j \cup \mathcal{P}$, and $[\tilde{\boldsymbol{z}}_j]_k = 0$ for $k \notin \mathcal{S} \cup \mathcal{P}$.
3: **for** $t = 1, 2, \ldots$ (until converged) **do**
4: Compute the set

$$\mathcal{A} := \bigcup_{j \in [B]} \{k \mid |[\nabla L(B(\tilde{\boldsymbol{z}}_j + \boldsymbol{\lambda}_j))]_k| > \tau_2\}.$$

5: Compute $[\boldsymbol{\lambda}_j^*]_k = \frac{1}{B}[\tilde{\boldsymbol{z}}]_k - [\tilde{\boldsymbol{z}}_j]_k$ for all $k \in \mathcal{A}$ and all $j \in [B]$.
6: Compute the step size $\eta = \frac{2}{t+2}$.
7: For all $k \in \mathcal{A}$ and all $j \in [B]$, update

$$[\boldsymbol{\lambda}_j]_k \leftarrow \eta[\boldsymbol{\lambda}_j^*]_k + (1-\eta)[\boldsymbol{\lambda}_j]_k.$$

8: **end for**
9: Output $\frac{1}{B}\sum_{j=1}^{B} \nabla L(B(\tilde{\boldsymbol{z}}_j + \boldsymbol{\lambda}_j))$.

3.2. Error Analysis

Define $\tilde{\boldsymbol{z}} = \sum_{j=1}^{B} \tilde{\boldsymbol{z}}_j$. Note that $\|\boldsymbol{z} - \tilde{\boldsymbol{z}}\|_\infty \leq B\tau_1$, so the error in approximating $L(\boldsymbol{z})$ by $L(\tilde{\boldsymbol{z}})$ is at most $B\tau_1$ times the ℓ_∞ Lipschitz constant of L, which is typically small as explained earlier. The algorithm essentially runs a Frank-Wolfe type method to converge to $L(\tilde{\boldsymbol{z}})$. In the following, we analyze the convergence rate of the greedy message passing algorithm (Algorithm 2) to $L(\tilde{\boldsymbol{z}})$. The analysis relies on smoothness of the loss function. A function is said to be $1/\mu$-smooth if its gradients are Lipschitz continuous with constant $1/\mu$. For the *Cross-Entropy loss* (2) we have $\mu = 1$, and for the *smoothed max-margin loss* (10), μ is a tunable parameter, and we found setting $\mu \in [1, 5]$ works well in our experiments.

To analyze the algorithm, denote by $\boldsymbol{\Lambda}$ the BK dimensional vector $\langle \boldsymbol{\lambda}_1, \boldsymbol{\lambda}_2, \ldots, \boldsymbol{\lambda}_B \rangle$ in any given step in the loop of the algorithm. Similarly let $\boldsymbol{\Lambda}^*$ denote the BK dimensional vector composed of $\boldsymbol{\lambda}_j^*$. Define $G(\boldsymbol{\Lambda}) = \frac{1}{B}\sum_{j=1}^{B} L(B(\tilde{\boldsymbol{z}}_j + \boldsymbol{\lambda}_j))$, i.e. the objective function in (12).

Theorem 2 (Greedy Message Passing)**.** *Suppose the loss function L is $1/\mu$-smooth. Then the suboptimality gap of $\boldsymbol{\Lambda}$ in the t-th step of the loop can be bounded as follows:*

$$G(\boldsymbol{\Lambda}) - G(\boldsymbol{\Lambda}^*) \leq \frac{2B\|\boldsymbol{\Lambda}^*\|^2}{\mu(t+2)} + 2\tau_2 \ln(t)\|\boldsymbol{\Lambda}^*\|_1$$

Proof. Since the loss function L is $1/\mu$-smooth, it is easy to check that G is B/μ-smooth. Thus, if $\Delta\boldsymbol{\Lambda}$ is the change in $\boldsymbol{\Lambda}$ in a given step of the loop in the algorithm, then

$$G(\boldsymbol{\Lambda} + \Delta\boldsymbol{\Lambda}) - G(\boldsymbol{\Lambda}) \leq \eta\langle \nabla G(\boldsymbol{\Lambda}), \Delta\boldsymbol{\Lambda} \rangle + \frac{\eta^2 B}{2\mu}\|\Delta\boldsymbol{\Lambda}\|^2.$$

Note that $\Delta\boldsymbol{\Lambda}$ equals $\boldsymbol{\Lambda}^* - \boldsymbol{\Lambda}$ in all coordinates except those corresponding to $k \notin \mathcal{A}$ for all $j \in [B]$, and the magnitude of the gradient in those coordinates is at most τ_2. Thus we have $\langle \nabla G(\boldsymbol{\Lambda}), \Delta\boldsymbol{\Lambda} \rangle \leq \langle \nabla G(\boldsymbol{\Lambda}), \boldsymbol{\Lambda}^* - \boldsymbol{\Lambda} \rangle + \tau_2\|\boldsymbol{\Lambda}^*\|_1$. Here, we used the fact that each coordinate of $\boldsymbol{\Lambda}$ lies between 0 and the corresponding coordinate of $\boldsymbol{\Lambda}^*$. Next, by the convexity of G, we have $\langle \nabla G(\boldsymbol{\Lambda}), \boldsymbol{\Lambda}^* - \boldsymbol{\Lambda} \rangle \leq G(\boldsymbol{\Lambda}^*) - G(\boldsymbol{\Lambda})$. Putting all the bounds together and following some algebraic manipulations, we have

$$\begin{aligned} &G(\boldsymbol{\Lambda} + \Delta\boldsymbol{\Lambda}) - G(\boldsymbol{\Lambda}^*) \\ &\leq (1-\eta)(G(\boldsymbol{\Lambda}) - G(\boldsymbol{\Lambda}^*)) + \eta\tau_2\|\boldsymbol{\Lambda}^*\|_1 + \frac{\eta^2 B}{2\mu}\|\boldsymbol{\Lambda}^*\|^2. \end{aligned} \tag{13}$$

Here, we used the fact that each coordinate of $\boldsymbol{\Lambda}$ lies between 0 and the corresponding coordinate of $\boldsymbol{\Lambda}^*$ to get the bound $\|\Delta\boldsymbol{\Lambda}\|^2 \leq \|\boldsymbol{\Lambda}^*\|^2$.

Now, using the fact that $\eta = \frac{2}{t+2}$ in iteration t, a simple induction on t implies the claimed bound on $G(\boldsymbol{\Lambda}) - G(\boldsymbol{\Lambda}^*)$.

□

Thus, to ensure that the suboptimality gap is at most ϵ, it suffices to run the greedy procedure for $T = \frac{B\|\boldsymbol{\Lambda}^*\|^2}{4\mu\epsilon}$ steps with $\tau_2 = \frac{\epsilon}{4\ln(T)\|\boldsymbol{\Lambda}^*\|_1}$. While this theorem provides a proof of convergence for the algorithm to any desired error level, the bound it provides is quite weak. In practice, we found that running just *one step* of the loop suffices to improve performance over direct search-based methods.

If, in addition to being smooth, the loss function is also strongly convex (which can be achieved by adding some ℓ_2^2 regularization, for instance) then we can also show convergence of the gradients. This is because for strongly convex functions the convergence of gradients can be bounded in terms of the convergence of the loss value. This is a very standard analysis and we omit it for the sake of clarity.

Cost Analysis. Exact gradient evaluation for a single sample can be computed in $O(DK)$ time. Directly applying a search-based gradient approximation (Algorithm 1) has a cost of $O(DQ_D(K))$, where $Q_D(K)$ is the number of classes retrieved in the MIPS data structure in order to find all classes of significant gradients. The query cost $Q_D(K)$ has a strong dependency on the dimension D. Exact MIPS has a cost $Q_D(K)$ exponential in D (Shrivastava & Li, 2014; Li & Malik, 2017). For approximate search methods,

such as Locality Sensitive Hashing (LSH), the cost $Q_D(K)$ typically only implicitly depends on the dimension. Our method (Algorithm 2) divides D into B subproblems of dimension D/B with a cost per message passing iteration of $O(DQ_{D/B}(K)+DB|\mathcal{A}|)$, where $\mathcal{A}$ is the set computed in step 4 of Algorithm 2. Note $Q_{D/B}(K)$ decreases with B rapidly (exponentially in the exact case) and therefore one can select B such that $Q_{D/B}(K) \ll Q_D(K)$ and balance two terms s.t. $(DQ_{D/B}(K) + DB|\mathcal{A}|) \ll DK$.

4. Practical Considerations

MIPS queries. In practice when using the MIPS data strcuctures, instead of retrieving all classes with scores more than the threshold τ_1, it is more efficient to retrieve the top Q classes with the highest scores. In our implementation, we use *Spherical Clustering* (Auvolat et al., 2015) as the MIPS data structure, where the number of clusters C is selected such that $K/C \leq Q$ and $C \leq Q$. Note this requires $Q \geq \sqrt{K}$, leading to a speedup bounded by $\sqrt{K}$. Similarly, for computing the active set $\mathcal{A}$ in step 4 of Algorithm 2, we can compute an appropriate threshold τ_2 using the properties of the loss function. In the case of margin-based losses, (3) and (7), and their smoothed versions (10), the gradient is sparse so τ_2 can be set to 0 or some very small value ($\tau_2 = 10^{-3}$ works well in our experiments). Loss functions like (2), (6) typically have exponentially decayed gradient magnitudes over the non-confusing negative classes. For these losses, classes can be retrieved in decreasing order of gradient magnitude, using a lower bound on the partition function $Z = \sum_k \exp z_k$ summing over only the subset of retrieved classes in order to decide whether more classes need to be retrieved or not.

Updates of data structures. During training the model parameters determining f will change, and the data structures $\mathcal{T}_j$ need to be updated. These data structures stores rows of $\boldsymbol{W}$ and treats $\boldsymbol{\phi}(\boldsymbol{x})$ as query. For loss functions with a sparse gradient, such as (3) and (7), and their smoothed versions (10), the number of updated rows of $\boldsymbol{W}$, k_r, is much smaller than K and Q (the number of classes retrieved for a query). Thus the cost for re-indexing rows of $\boldsymbol{W}$ is $k_r C(D/B)B = k_r CD$, where C is the number of inner products required to index each row, which is much smaller than the costs of query and updates. For tasks with large number of updated rows ($k_r \approx Q$), the method is still effective with a larger mini-batch size N_b. As the costs of query and updates grow with N_b while the number of rows to re-index is bounded by K, the cost of maintaining data structure becomes insignificant.

Sampling for initialization. For a randomly initialized model, the early iterates of learning have gradients evenly distributed over the classes, as the scores of all classes are close to each other. Therefore, it is unnecessary to search candidates of significant gradient magnitude in the early stage. In practice, one can switch from a sampling-based gradient approximation to a search-based gradient approximation after a number of mini-batch updates. In our experiments of unsupervised learning of word embeddings, we initialize the algorithm with a single epoch of SGD with sampling gradient approximation.

5. Experiments

In this section, we conduct experiments on three types of problems: (i) multiclass classification (face recognition), (ii) multilabel classification (document tagging), and (iii) Unsupervised Word Embedding (Skip-gram objective (9)). For multiclass and multilabel classification, we employ a Stochastic Gradient Descent (SGD) optimization algorithm, with an initial step size chosen from $\{1, 0.1, 0.01\}$ for the best performance of each method, with a $1/(1+t)$ cooling scheme where t is the iteration counter. The mini-batch size is 10 and all methods are parallelized with 10 CPU cores in a shared-memory architecture, running on a dedicated machine. All the implementation are in C++. The following loss functions and gradient evaluation methods are compared for the experiments on multiclass and multilabel classification:

- **Softmax**: exact gradient evaluation of the cross-entropy loss (2). For multiclass, we have $|\mathcal{P}| = 1$ and for multilabel, $|\mathcal{P}| \ll K$.
- **Sampled-Softmax**: the sampling strategy in (Jean et al., 2014; Chen et al., 2015), which includes all positive classes of the instances and uniformly subsamples from the remaining negative classes. Here we choose sample size as $K/100$.
- **Margin**: exact gradient evaluation of the *smoothed max-margin loss* (10), where we choose $\mu = 1$ for the case of multiclass, and $\mu = 5$ for the case of multilabel. The bi-simplex projection (11) is computed in $O(K \log K)$ using the procedure described in (Yen et al., 2016). Note the gradient update for this loss is faster than that for *cross-entropy*, as the loss gradient is very sparse, making the backward pass much faster.
- **MIPS**: search-based gradient evaluation (Algorithm 1) with *smoothed max-margin loss* (same setting to **Margin**). We use *Spherical Clustering* (Auvolat et al., 2015) with 100 centroids as the MIPS data structure, and a batch query of size $K/100$.
- **Decomp-MIPS**: gradient evaluation via decomposed search (Algorithm 2, $T = 1$ iteration). We divide the inner product into $B = 8$ factors in the multiclass experiment and $B = 4$ in the multilabel case. The settings for MIPS data structure are the same as above.

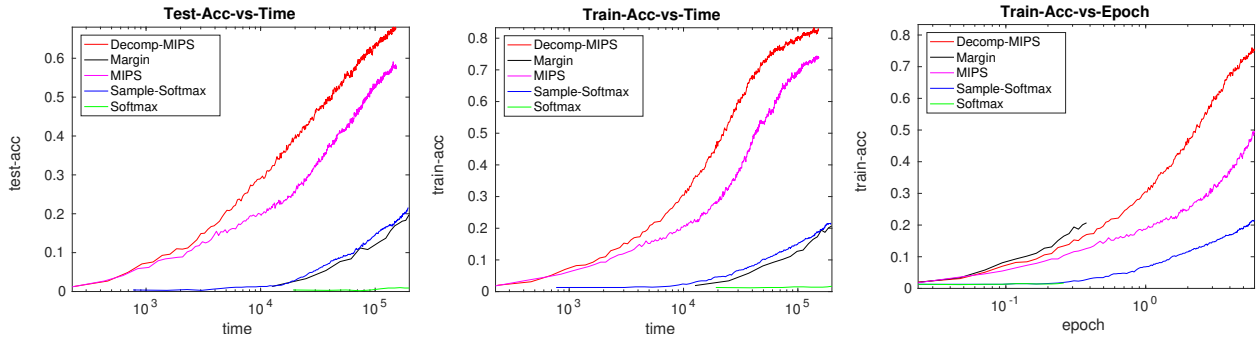

Figure 1. Results of multiclass classification on the *Megaface* data set: Test Accuracy vs. Training time (left), Test Accuracy vs. Training Time (middle), and Training Accuracy vs. number of epochs (right). Note the x-axis is in log-scale, and the curves are actually close to convergence when plotted in linear scale.

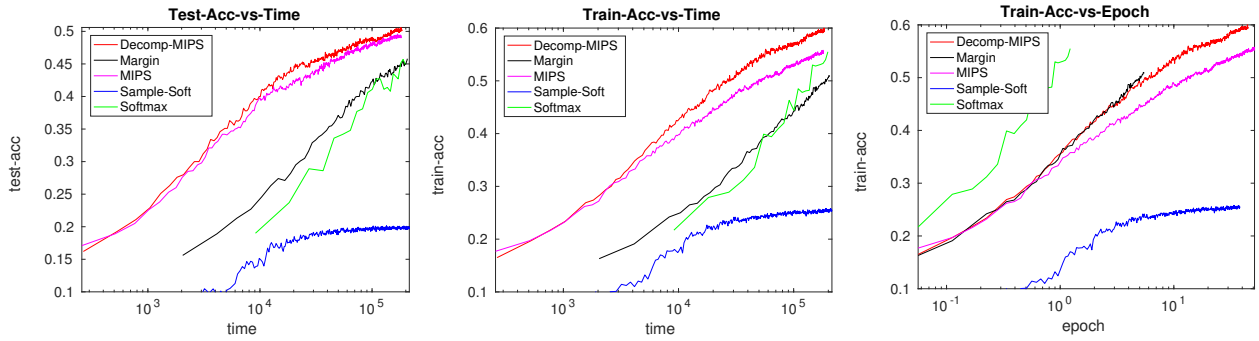

Figure 2. Results of multilabel classification on the *WikiLSHTC* data set: Test Accuracy vs. Training time (left), Training Accuracy vs. Training Time (middle), and Training Accuracy vs. number of epochs (right). Note the x-axis is in log-scale, and the curves are actually close to convergence when plotted in linear scale.

5.1. Multiclass Classificatoin

#Identities	#Images	Embed. Dim.
672K	4.7M	128

Table 1. Statistics of the MegaFace dataset.

For multiclass classification we conduct experiments on the largest publicly available facial recognition dataset *MegaFace* (Challenge 2)[2], where each identity is considered a class, and each sample is an image cropped by a face detector. The data set statistics are shown in Table 1.

We employ the *FaceNet* architecture (Schroff et al., 2015)[3] pre-trained on the *MS-Celeb-1M* dataset, and fine-tune its last layer on the *MegaFace* dataset. The input of the last layer is an embedding of size 128, which is divided into $B = 8$ factors, each of dimension 16, in the *Decomp-MIPS* method.

The result is shown in Figure 1, where all methods are run for more than one day. Firstly, comparing methods that optimize the (smoothed) max-margin loss (*Decomp-MIPS*, *MIPS* and *Margin*) shows that both *Decomp-MIPS*, *MIPS* speed up the iterates by $1 \sim 2$ orders of magnitude. However, *MIPS* converges at an accuracy much lower than *Decomp-MIPS* and the gap gets bigger when running for more iterations. Note the time and epochs are in log scale. Secondly, *Softmax* has a much slower progress compared to *Margin*. Note both of them do not even finish one epoch (4.7M samples) after one day, while the progress of *Margin* is much better, presumably because its focus on the *confusing identities*. *Sampled-Softmax* has much faster iterates, but the progress per iterate is small, leading to slower overall progress compared to the MIPS-based approaches.

5.2. Multilabel Classification

For multilabel classification, we conduct experiments on *WikiLSHTC* (Partalas et al., 2015), a benchmark data set in the Extreme Classification Repository[4], where each class is a catalog tag in the Wikipedia, and each sample is a document with bag of words representation. The data statistics

[2] http://megaface.cs.washington.edu/.
[3] github.com/davidsandberg/facenet
[4] manikvarma.org/downloads/XC/XMLRepository.html

are shown in Table 2.

We train a one-hidden-layer fully-connected feedforward network for the multilabel classification task. The first layer has input dimension equal to the vocabulary size (1.6M) and an output of dimension 100. The second layer has output size equal to the number of classes (325K), with different loss functions and approximations for different methods in comparison. The training result also produces document and work embedding as by-products. For *Decomp-MIPS*, the input of the last layer is divided into $B = 4$ factors, each of dimension 25.

We run all the compared methods for more than one day and the result is shown in Figure 2. First, for this multilabel task, *Softmax* has very good per-iteration progress, significantly more than that from the other three approaches based on the smoothed max-margin loss (*Margin*, *MIPS*, *Decomp-MIPS*). However, the iterates of *Softmax* are much slower than the others as it has a dense loss gradient and thus a slower backpropagation, so that when comparing training time, *Softmax* performs similarly to *Margin*. On the other hand, when comparing *Margin Decomp-MIPS*, and *MIPS* in progress per epoch, the updates of *Decomp-MIPS* achieve almost the same progress as the exact gradient calculation of *Margin*, while *MIPS* has a significant drop in its training accuracy compared with *Margin* and *Decomp-MIPS*, since it runs for more iterations. Overall, the *MIPS*-based methods lead to an order of magnitude speedup, while *Decomp-MIPS* retains the accuracy of the exact method. On the other hand, *Sampled-Softmax* has an extremely slow per-iteration progress despite its fast iterates, and could not reach a comparable accuracy to other methods even after one day.

#Label	#Sample	Embed. Dim.	Vocab. Size
325K	1.8M	100	1.6M

Table 2. Statistics of the *WikiLSHTC* data set. On average, each sample has 3.19 positive labels, and each class appears in 17.46 samples as a positive class.

5.3. Unsupervised Word Embedding

Vocab. Size	#Words	Embed. Dim.	Window Size
$\approx$ 451K	$\approx$ 680M	100	8

Table 3. Statistics of the BillionW dataset.

In this section, we evaluate the proposed gradient approximation technique on the word embedding task with the *Skip-gram* learning objective (9) and compare it with two widely-used gradient approximation methods — *Hierarchical Softmax (Word2vec-HS)* and *Negative Sampling (Word2vec-Neg)* (Mikolov et al., 2013) implemented in the *word2vec*[5] package released by the authors. The sample size for *Word2vec-Neg* is selected from $\{5, 10, 15, 20, 25\}$.

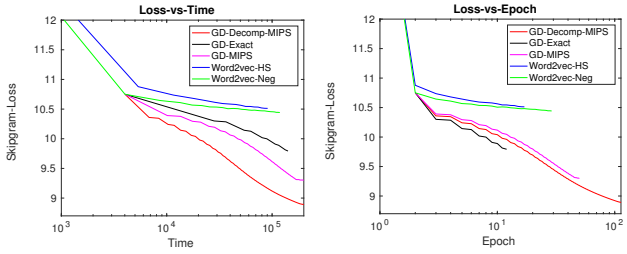

Figure 3. Results on word embedding with *Skip-gram* objective, where *GD-Exact*, *GD-MIPS*, and *GD-Decomp-MIPS* are initialized with a model trained by one epoch of *Word2vec-Neg*.

We use the benchmark data set *BillonW*[6] of almost a half million vocabulary size. The data statistics are provided in Table 3. Following (Mikolov et al., 2013), we use a window of size 8 and subsample *frequent words* in the corpus. Each word w is dropped with probability $\max\{1 - \sqrt{\frac{t}{f_w}}, 0\}$ where f_w is the relative frequency of the word in the corpus, and $t = 10^{-4}$ is a threshold parameter.

Note that the *Skip-gram* objective (9) is presented in a collapsed form equivalent to the one in (Mikolov et al., 2013). Here, all terms of the same input-output pairs are grouped together and weighted by the frequency. We compute gradients from the positive outputs by summing over the empirical input-output distribution $q_{\boldsymbol{x}}$, $q_{\boldsymbol{y}|\boldsymbol{x}}$ in (9). Then we perform gradient descent (GD) updates on the parameters of input words $\{\boldsymbol{\phi}(\boldsymbol{x})\}_{\boldsymbol{x}\in\mathcal{X}}$ and output words $\{\boldsymbol{\phi}(\boldsymbol{y})\}_{\boldsymbol{y}\in\mathcal{X}}$ alternately. We use *GD*, *GD-MIPS* and *GD-Decomp-MIPS* to denote the algorithm with different strategies of loss approximations. As mentioned in Section 4, since in the early iterates the model has quite evenly distributed gradient over candidates, we use 1 epoch of *Word2vec-Neg* to initialize *GD*, *GD-MIPS* and *GD-Decomp-MIPS*. For this task, we have many more negative classes of significant gradient magnitude than in the multilabel and multiclass experiments. So we use a batch query of size $K/20$ instead of $K/100$ to the MIPS structure. All the compared methods are parallelized with 24 CPU cores.

The results are shown in Figure 3. After the first epoch, methods based on alternating gradient descent (GD) (with the collapsed objective (9)) have faster convergence per epoch, and the iterations of *GD-Deomp-MIPS* are 5 times faster than those of *GD* while having a significantly better objective value than *GD-MIPS* for the same training time.

[5] code.google.com/archive/p/word2vec/

[6] www.statmt.org/lm-benchmark/

References

Auvolat, Alex, Chandar, Sarath, Vincent, Pascal, Larochelle, Hugo, and Bengio, Yoshua. Clustering is efficient for approximate maximum inner product search. *arXiv preprint arXiv:1507.05910*, 2015.

Bhatia, Kush, Jain, Himanshu, Kar, Purushottam, Varma, Manik, and Jain, Prateek. Sparse local embeddings for extreme multi-label classification. In *Advances in Neural Information Processing Systems*, pp. 730–738, 2015.

Boyd, Stephen, Parikh, Neal, Chu, Eric, Peleato, Borja, Eckstein, Jonathan, et al. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends in Machine Learning*, 3(1):1–122, 2011.

Chen, Welin, Grangier, David, and Auli, Michael. Strategies for training large vocabulary neural language models. *arXiv preprint arXiv:1512.04906*, 2015.

Goldberger, Jacob, Hinton, Geoffrey E, Roweis, Sam T, and Salakhutdinov, Ruslan R. Neighbourhood components analysis. In *Advances in neural information processing systems*, pp. 513–520, 2005.

Guo, Ruiqi, Kumar, Sanjiv, Choromanski, Krzysztof, and Simcha, David. Quantization based fast inner product search. In *Artificial Intelligence and Statistics*, pp. 482–490, 2016.

Gutmann, Michael U and Hyvärinen, Aapo. Noise-contrastive estimation of unnormalized statistical models, with applications to natural image statistics. *Journal of Machine Learning Research*, 13(Feb):307–361, 2012.

Jean, Sébastien, Cho, Kyunghyun, Memisevic, Roland, and Bengio, Yoshua. On using very large target vocabulary for neural machine translation. *arXiv preprint arXiv:1412.2007*, 2014.

Li, Ke and Malik, Jitendra. Fast k-nearest neighbour search via prioritized dci. In *International Conference on Machine Learning*, pp. 2081–2090, 2017.

Mikolov, Tomas, Sutskever, Ilya, Chen, Kai, Corrado, Greg S, and Dean, Jeff. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pp. 3111–3119, 2013.

Mnih, Andriy and Hinton, Geoffrey E. A scalable hierarchical distributed language model. In *Advances in neural information processing systems*, pp. 1081–1088, 2009.

Morin, Frederic and Bengio, Yoshua. Hierarchical probabilistic neural network language model. In *Aistats*, volume 5, pp. 246–252. Citeseer, 2005.

Movshovitz-Attias, Yair, Toshev, Alexander, Leung, Thomas K, Ioffe, Sergey, and Singh, Saurabh. No fuss distance metric learning using proxies. *arXiv preprint arXiv:1703.07464*, 2017.

Mussmann, Stephen and Ermon, Stefano. Learning and inference via maximum inner product search. In *International Conference on Machine Learning*, pp. 2587–2596, 2016.

Mussmann, Stephen, Levy, Daniel, and Ermon, Stefano. Fast amortized inference and learning in log-linear models with randomly perturbed nearest neighbor search. *arXiv preprint arXiv:1707.03372*, 2017.

Nesterov, Yu. Smooth minimization of non-smooth functions. *Mathematical programming*, 103(1):127–152, 2005.

Partalas, Ioannis, Kosmopoulos, Aris, Baskiotis, Nicolas, Artieres, Thierry, Paliouras, George, Gaussier, Eric, Androutsopoulos, Ion, Amini, Massih-Reza, and Galinari, Patrick. Lshtc: A benchmark for large-scale text classification. *arXiv preprint arXiv:1503.08581*, 2015.

Prabhu, Yashoteja and Varma, Manik. Fastxml: A fast, accurate and stable tree-classifier for extreme multi-label learning. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 263–272. ACM, 2014.

Schroff, Florian, Kalenichenko, Dmitry, and Philbin, James. Facenet: A unified embedding for face recognition and clustering. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 815–823, 2015.

Shrivastava, Anshumali and Li, Ping. Asymmetric lsh (alsh) for sublinear time maximum inner product search (mips). In *Advances in Neural Information Processing Systems*, pp. 2321–2329, 2014.

Spring, Ryan and Shrivastava, Anshumali. A new unbiased and efficient class of lsh-based samplers and estimators for partition function computation in log-linear models. *arXiv preprint arXiv:1703.05160*, 2017a.

Spring, Ryan and Shrivastava, Anshumali. Scalable and sustainable deep learning via randomized hashing. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 445–454. ACM, 2017b.

Vijayanarasimhan, Sudheendra, Shlens, Jonathon, Monga, Rajat, and Yagnik, Jay. Deep networks with large output spaces. *arXiv preprint arXiv:1412.7479*, 2014.

Weinberger, Kilian Q and Saul, Lawrence K. Distance metric learning for large margin nearest neighbor classification. *Journal of Machine Learning Research*, 10(Feb): 207–244, 2009.

Wu, Xiang, Guo, Ruiqi, Suresh, Ananda Theertha, Kumar, Sanjiv, Holtmann-Rice, Daniel N, Simcha, David, and Felix, X Yu. Multiscale quantization for fast similarity search. In *Advances in Neural Information Processing Systems*, pp. 5749–5757, 2017.

Yen, Ian EH, Huang, Xiangru, Dai, Wei, Ravikumar, Pradeep, Dhillon, Inderjit, and Xing, Eric. Ppdsparse: A parallel primal-dual sparse method for extreme classification. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 545–553. ACM, 2017.

Yen, Ian En-Hsu, Chang, Chun-Fu, Lin, Ting-Wei, Lin, Shan-Wei, and Lin, Shou-De. Indexed block coordinate descent for large-scale linear classification with limited memory. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 248–256. ACM, 2013.

Yen, Ian En-Hsu, Huang, Xiangru, Ravikumar, Pradeep, Zhong, Kai, and Dhillon, Inderjit. Pd-sparse: A primal and dual sparse approach to extreme multiclass and multilabel classification. In *International Conference on Machine Learning*, pp. 3069–3077, 2016.

Yu, Hsiang-Fu, Jain, Prateek, Kar, Purushottam, and Dhillon, Inderjit. Large-scale multi-label learning with missing labels. In *International conference on machine learning*, pp. 593–601, 2014.