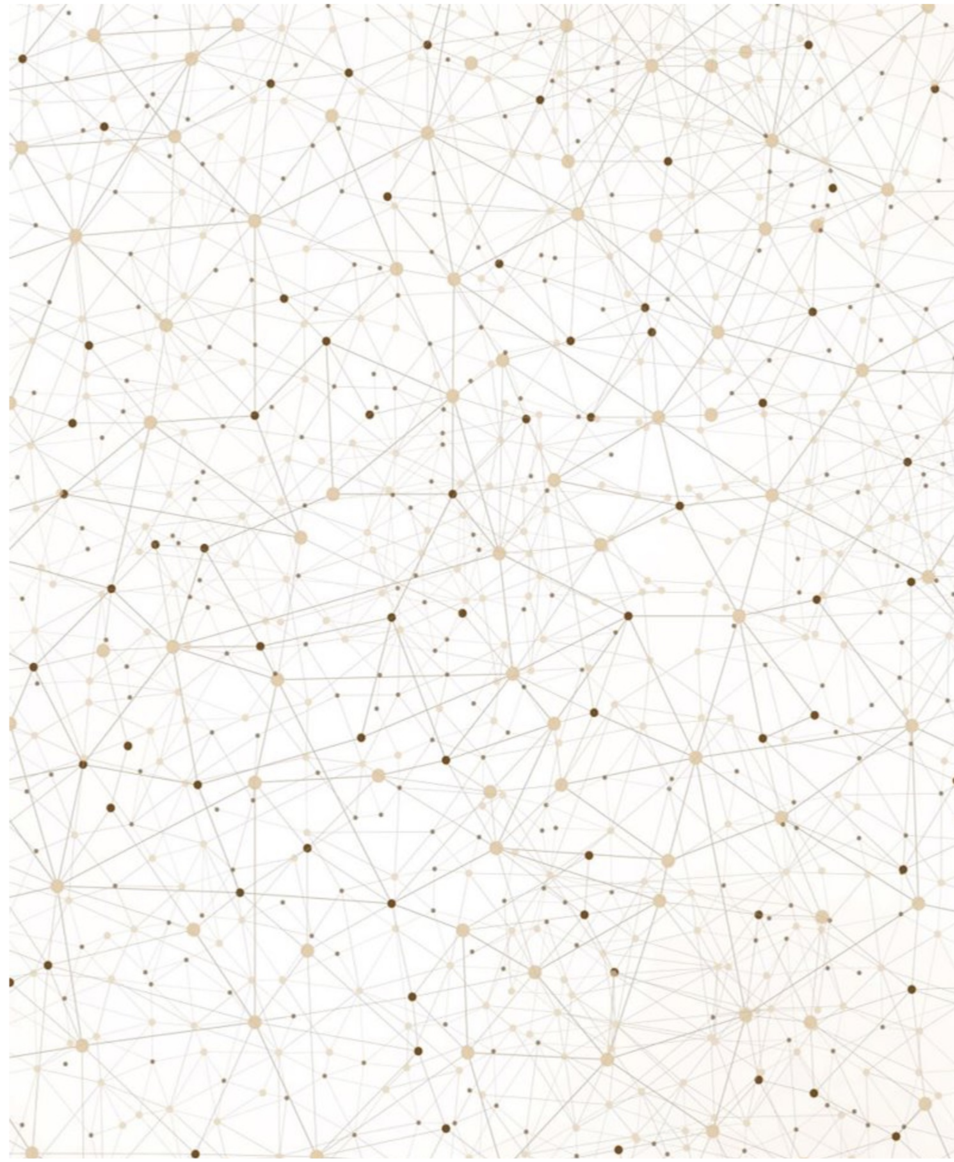
Global Selection of Contrastive Batches via Optimization on Sample Permutations

Vin Sachidananda[1] , Ziyi Yang[2], Chenguang Zhu[2]

[1]Stanford University and Two Sigma Ventures
[2]Knowledge and Language Team, Microsoft Research

ICML 2023



(I) CONTRASTIVE LEARNING

- **Contrastive Learning**, used in many tasks for aligning embeddings in unimodal and multimodal tasks
  - Sentence Embeddings[1]
  - Code-Language Models[2]
  - Language-Image Models[3]

In this work, improve SOTA on both these models.

- All using same **NT-Xent objective** (scaled normalized cross entropy) to:
  - **Maximize inner product** of **similar data**
  - **Minimize inner product** of **other data**

[1](SimCSE: Gao 2021, DCPCSE: Jiang 2022) [2](UniXcoder: Guo, 2022) [3](CLIP: Radford 2021, CoCa: Yu 2022)

(I) CONTRASTIVE LEARNING

Supervised contrastive learning, pair of datasets X, Y of size N.

- $X_i \approx Y_i$
- $X_i \neq Y_j, \forall i \neq j$

Example datasets include:

- pairs of similar sentences
- images and text captions
- code and their comments

```
def _parse_memory(s):
    """
    Parse a memory string in the format supported by Java (e.g. 1g, 200m) and
    return the value in MiB

    >>> _parse_memory("256m")
    256
    >>> _parse_memory("2g")
    2048
    """
    units = {'g': 1024, 'm': 1, 't': 1 << 20, 'k': 1.0 / 1024}
    if s[-1].lower() not in units:
        raise ValueError("invalid format: " + s)
    return int(float(s[:-1]) * units[s[-1].lower()])
```

*Image from CodeBERT (Feng, 2020)

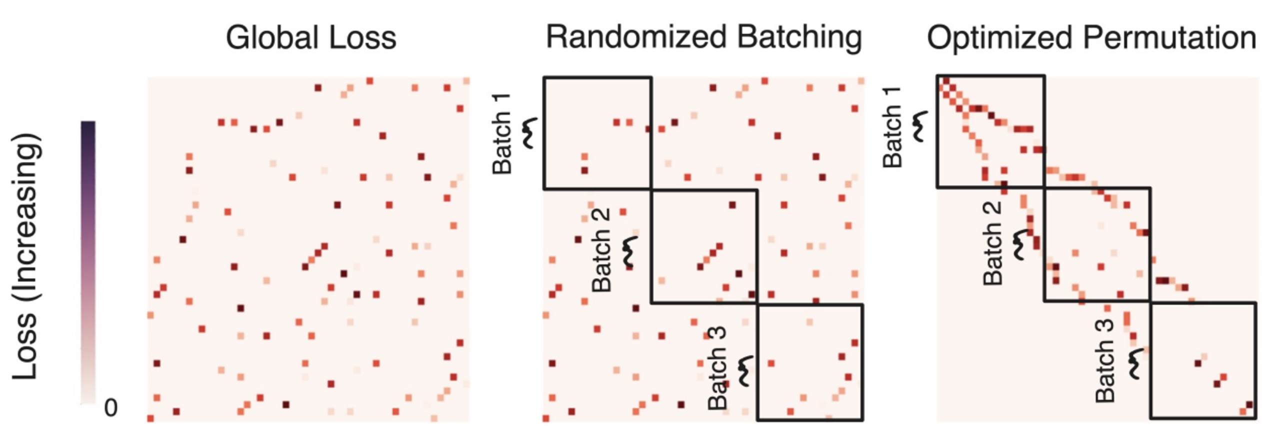
(I) CONTRASTIVE LEARNING

- Batch B, of $\boldsymbol{k}$ **samples**, drawn without replacement, from both X, Y.
- **Maximize** $f(X_i)^T f(Y_i)$ and **minimize** $f(X_i)^T f(Y_j)\ \forall j \neq i$ for $j \in B_i$.
- Use scaled cross entropy loss (only comparing in-batch inner products):

$$\mathcal{L}_i = -\log \frac{\exp(f(X_i)^T f(Y_i)\tau^{-1})}{\sum_{j \in B_i} \exp(f(X_i)^T f(Y_j)\tau^{-1})}$$

(II) GAP BETWEEN GLOBAL AND OBSERVED LOSSES

- Can only *observe* Nk *out of* N^2 *inner products* during each training epoch. *Which Nk to use?*



Goal: Minimize the gap between total and observed loss by permuting over rows of X, Y.

(II) GAP BETWEEN GLOBAL AND OBSERVED LOSSES

- Training (observed) loss is poor approximation for global loss if large inner product values of, $x_i^T y_j$, not drawn in batches.

$$\mathcal{L}^{Global} - \mathcal{L}^{Train} = \frac{1}{N}\sum_{i=1}^{N} \log \sum_{j=1}^{N} \exp(x_i^T y_j \tau^{-1}) - \log \sum_{j \in B_i} \exp(x_i^T y_j \tau^{-1}) = \frac{1}{N}\sum_{i=1}^{N} \log \frac{\sum_{j=1}^{N} \exp(x_i^T y_j \tau^{-1})}{\sum_{j \in B_i} \exp(x_i^T y_j \tau^{-1})}$$

- Increasing value of inner products in batches reduces $L^{global} - L^{train}$. Observed in prior works on hard negative mining.[1]

[1](Zhang et al., 2018; Xiong et al., 2021)

(III) BOUNDS ON THE GAP BETWEEN LOSSES

- By using Log-Sum-Exp bounds gap can be minimized as either a Quadratic Assignment Problem or Quadratic Bottleneck Assignment Problem.[1] **(Both NP-Hard)**

Theorem 4.1 (Formulation of QBAP for bound in Theorem 3.6). *The following Quadratic Bottleneck Assignment Problem, minimizes the upper bound provided in Theorem 3.6 summed over X and Y:*

$$\boxed{\min_{\pi \in \Pi_N} \max_{i,j} -A \odot \pi Z \pi^T.}$$

Theorem 4.2 (Formulation of QAP for bound in Theorem 3.7). *The following Quadratic Assignment Problem minimizes the upper bound in Theorem 3.7:*

$$\boxed{\max_{\pi \in \Pi_N} Tr(A\pi(XY^T + YX^T)\pi^T).}$$

[1](Koopmans and Beckman, 1957)

(IV) EFFICIENT APPROXIMATION TO THE QBAP (GCBS)

- First, sparsify XY^T on large quantile (i.e. keep largest Nk inner products).

$$(\tilde{XY^T})_{i,j} = \begin{cases} 1, & x_i^T y_j > q, i \neq j \\ 0, & else. \end{cases}$$

- Relax QBAP to Matrix Bandwidth Minimization, Cuthill-Mckee heuristic returns permutation $\pi \in \Pi_N$
  - **time complexity:** $O(\text{Nm} \log(\text{m})) \leq O(N^2 \log(N))$, m is max degree
  - **space complexity:** $O(\text{Nk})$
  - **implementation:** 15 lines of PyTorch

- Reorder data as πX and πY, use a SequentialSampler to get batches during training.

(V) EXPERIMENTATION: CODESEARCH

- Using GCBS, we achieve state of the art results on joint Code-Language Embeddings as evaluated on code search task sets (CosQA, AdvTest, CSN) **improving SOTA MRR (x100) by ~2.2**.

Model	CosQA	AdvTest	Ruby	JS	Go	Python	Java	PHP	CSN Avg
RoBERTa	60.3	18.3	58.7	51.7	85.0	58.7	59.9	56.0	61.7
CodeBERT	65.7	27.2	67.9	62.0	88.2	67.2	67.6	62.8	69.3
GraphCodeBERT	68.4	35.2	70.3	64.4	89.7	69.2	69.1	64.9	71.3
SYNCoBERT	-	38.3	72.2	67.7	91.3	72.4	72.3	67.8	74.0
PLBART	65.0	34.7	67.5	61.6	88.7	66.3	66.3	61.1	68.5
CodeT5-base	67.8	39.3	71.9	65.5	88.8	69.8	68.6	64.5	71.5
UniXcoder	70.1	41.3	74.0	68.4	91.5	72.0	72.6	67.6	74.4
- with GCBS	**71.1** (+1.0)	**43.3** (+2.0)	**76.7**	**70.6**	**92.4**	**74.6**	**75.3**	**70.2**	**76.6** (**+2.2**)

Table 1. The performance comparison of supervised models along with a comparison of the best performing model (UniXcoder) (Guo et al., 2022) when using GCBS vs the standard Random Sampling. The reported score is Mean Reciprical Rank magnified by a factor of 100. GCBS improves previous best MRR when used with UniXcoder by 2.2 points achieving new state-of-the-art results (Row shaded gray).

(V) EXPERIMENTATION: SENTENCE EMBEDDING

- Using GCBS, we achieve **state of the art results on Sentence Embeddings** as evaluated on STS.
- Performance gain in Spearman Correlation for SimCSE and DCPCSE Roberta-Large is **+1.03%** and **+0.37%** respectively.

Model	STS12	STS13	STS14	STS15	STS16	STS-B	SICK-R	Avg
SBERT$_{base}$	70.97	76.53	73.19	79.09	74.30	77.03	72.91	74.89
SBERT$_{base}$-flow	69.78	77.27	74.35	82.01	77.46	79.12	76.21	76.60
SBERT$_{base}$-whitening	69.65	77.57	74.66	82.27	78.39	79.52	76.91	77.00
ConSERT-BERT$_{base}$	74.07	83.93	77.05	83.66	78.76	81.36	76.77	79.37
SimCSE-BERT$_{base}$	75.30	84.67	80.19	85.40	80.82	84.25	80.39	81.57
- with GCBS	75.81	85.30	81.12	86.58	81.68	84.80	80.04	82.19 (+0.62)
PromCSE-BERT$_{base}$	75.96	84.99	80.44	86.83	81.30	84.40	80.96	82.13
- with GCBS	75.20	85.00	81.00	86.82	82.55	84.76	79.95	82.18 (+0.05)
SimCSE-RoBERTa$_{base}$	76.53	85.21	80.95	86.03	82.57	85.83	80.50	82.52
- with GCBS	76.94	85.64	81.87	86.84	82.78	85.87	80.68	82.95 (+0.43)
PromCSE-RoBERTa$_{base}$	77.51	86.15	81.59	86.92	83.81	86.35	80.49	83.26
- with GCBS	77.33	86.77	82.19	87.57	84.09	86.78	80.05	83.54 (+0.28)
SimCSE-RoBERTa$_{large}$	77.46	87.27	82.36	86.66	83.93	86.70	81.95	83.76
- with GCBS	78.90	88.39	84.18	88.32	84.85	87.65	81.27	84.79 (+1.03)
PromCSE-RoBERTa$_{large}$	79.56	88.97	83.81	88.08	84.96	**87.87**	**82.43**	85.10
- with GCBS	**80.49**	**89.17**	**84.57**	**88.61**	**85.38**	**87.87**	81.49	**85.37** (**+0.27**)

Table 2. The performance comparison of supervised models along with a comparison of the best performing models, SimCSE (Gao et al., 2021) and PromCSE (Yuxin Jiang & Wang, 2022), with and without GCBS. The reported score is Spearman correlation magnified by a factor of 100. For RoBERTa$_{large}$ backbone models, GCBS improves previous best Spearman correlation when used with SimCSE by 1.03 points and PromCSE by 0.27 points achieving new state-of-the-art results.

(VI) DISCUSSION

- Empirically, **GCBS reduces the gap between the global and expected observed loss during training by 40%** for the Code Search Net (Ruby) dataset with the UniXcoder model.
- At the 10th epoch, the **per sample observed loss with GCBS is 15x larger** than that of Random Sampling.
- At the 10th epoch, the **global loss per sample is 30% less** when using GCBS than that of Random Sampling.

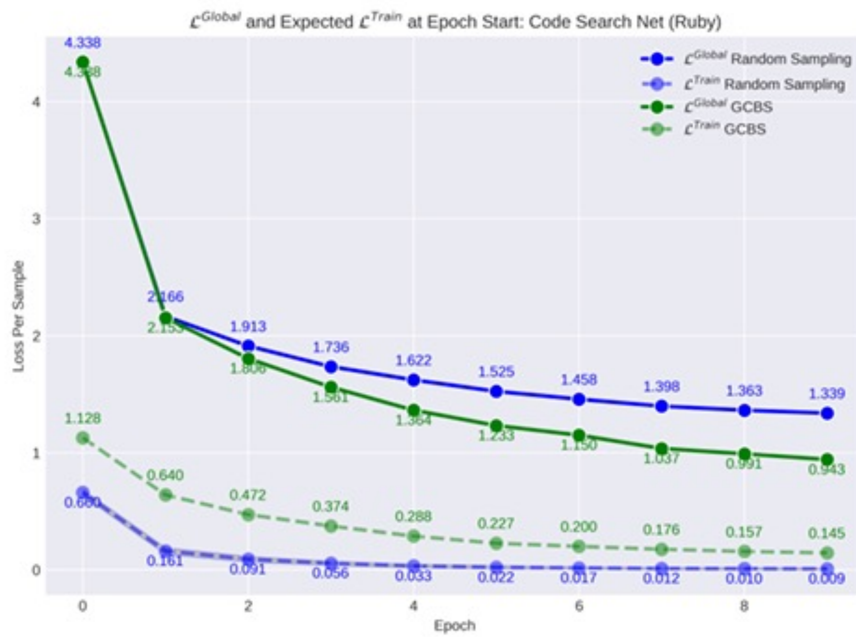


Figure 3. $\mathcal{L}^{Global}$ and Expected $\mathcal{L}^{Train}$ at the start of each epoch for Random Sampling and GCBS on the Code Search Net (Ruby) dataset with the UniXcoder model.

(VI) DISCUSSION

- GCBS takes ~8.5 minutes to run across 275K contrastive pairs and is more efficient than current global approaches for batch assignment (hard negative mining).

	Code Search		
Step	Random	GCBS	Hard Negative (1)
Fwd+Bkwd Pass	381.45	381.45	762.9 (2x batches)
Add'l Fwd Pass	-	118.51	118.51
Comp. k-NN	-	-	1.19
GCBS	-	2.31	-
Total Time (s)	381.45	502.27	882.61

Table 5. Runtime in seconds per epoch for Random Sampling, GCBS, and Hard Negative (1) for the Code Search Net (Ruby) dataset $N = 24,927, k = 64$ with the UniXcoder model.

	Sentence Embedding		
Step	Random	GCBS	Hard Negative (1)
Fwd+Bkwd Pass	442.26	442.26	884.52 (2x batches)
Add'l Fwd Pass	-	370.31	370.31
Comp. k-NN	-	-	225.99
GCBS	-	140.32	-
Total Time (s)	442.26	965.03	1480.82

Table 6. Runtime in seconds per epoch for Random Sampling, GCBS, and Hard Negative (1) for the SNLI+MNLI (entailment+hard neg) dataset $N = 275,602, k = 256$ for sentence embedding with the Bert-base-uncased model.

(VII) IMPLEMENTATION IN PYTORCH

- The Global Contrastive Batch Sampling method pseudocode in PyTorch is shown below. Full code is available at https://github.com/vinayak1/GCBS

```python
def compute_perm_bandwidth_min(X, Y, quantile_thresh = 0.999):
    # (1) Normalize representations.
    X, Y = normalize(X), normalize(Y)

    # (2) Get value at quantile threshold on the inner product matrix.
    quantile_thresh = torch.quantile(X @ Y.T, quantile_thresh)

    # (3) Get inner product matrix hard thresholded on quantile.
    row, col, data = [], [], []

    # Get rows and columns of indices > estimated quantile value
    ret = ((X @ Y.T).flatten() > quantile_thresh).nonzero
    row += ((ret - (ret % num_samples))/num_samples).tolist()
    col += (ret % num_samples).tolist()
    data += [1.0 for _ in range(len(ret))]

    # (4) Get perm which minimizes bandwidth of sparsified matrix with Cuthill-McKee.
    permutation = list(cuthill_mckee(sparse_matrix((data, (row, col)),
                                     shape=(num_samples, num_samples))))
    return permutation
```