Sigmoid Loss for Language Image Pre-Training

Zhai et al., ICCV 2023

2024.8.14

Lab Seminar

Harim Noh

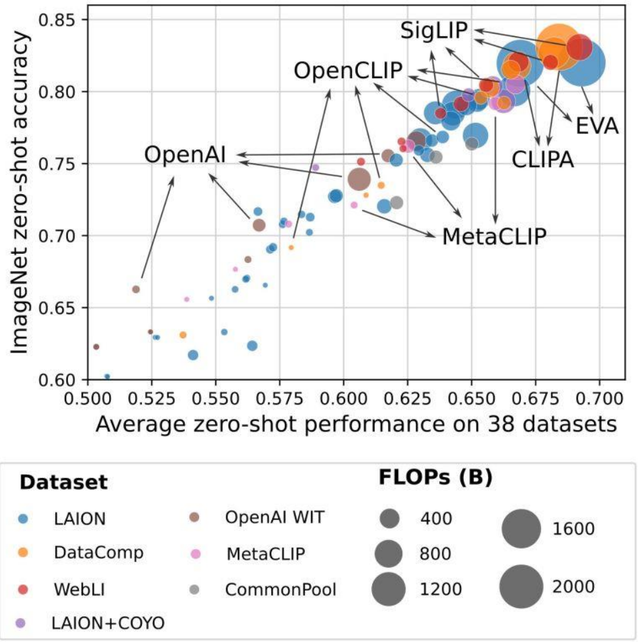
Duksung Women's University
Dept. of Computer Engineering

Overview

- Contrastive Learning에서 softmax loss를 **sigmoid loss**로 교체

  - softmax 기반 손실 함수에 비해 메모리와 계산 효율성을 크게 개선

  - chunk 방식의 구현을 통해 대규모 배치 크기에서도 효율적인 학습 가능



[Zero-shot accuracy]

Introduction

- **Sigmoid loss**

  - 기존의 softmax loss는 이미지-텍스트 pair의 유사도를 계산하기 위해 전체 배치에서 2번의 정규화 요구

    ➔ 불안정하고 계산 비용이 많이 들게 됨

  - Sigmoid loss는 전체 배치에 대한 작업을 필요하지 않음

  - CLIP과 LiT 같은 주요 이미지-텍스트 학습 모델에 적용하여 성능을 비교

➔ SigLIP & SigLiT

Background

- **Sigmoid loss**

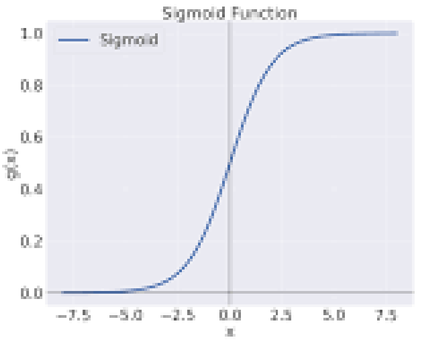
  - 이진 분류 문제와 쌍 간의 유사도를 비교 문제에 사용

  - 각 쌍의 유사도를 **개별적으로 처리함**

  - 모델의 유사도 점수가 실제 레이블(긍정/부정)과 얼마나 일치하는지를 측정

➔ 각 쌍을 **독립적으로** 평가하고 계산이 간단하여 메모리와 계산 효율성이 높지만, 전체 배치에서의 쌍 간 관계를 충분히 반영하지 못할 수 있음

$$g(x) = \frac{1}{1+e^{-x}} \in (0, 1)$$



[sigmoid function]

Background

- **Softmax loss**

  - 다중 클래스 분류 문제에 사용
  - 각 **클래스에 대해** 확률을 예측, 전체 클래스의 점수에 대한 상대적 우위를 평가
  - 예측된 확률과 실제 레이블 간의 차이를 측정하여 모델의 예측을 개선

➔ **클래스 간의 관계를** 고려하여 확률 분포를 생성하며, 대규모 데이터셋에서 계산 비용이 크고 수치적으로 불안함

$$\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_j e^{z_j}} \qquad L_{\text{softmax}} = -\sum_i y_i \log(\text{softmax}(z_i))$$

[softmax function]

Background

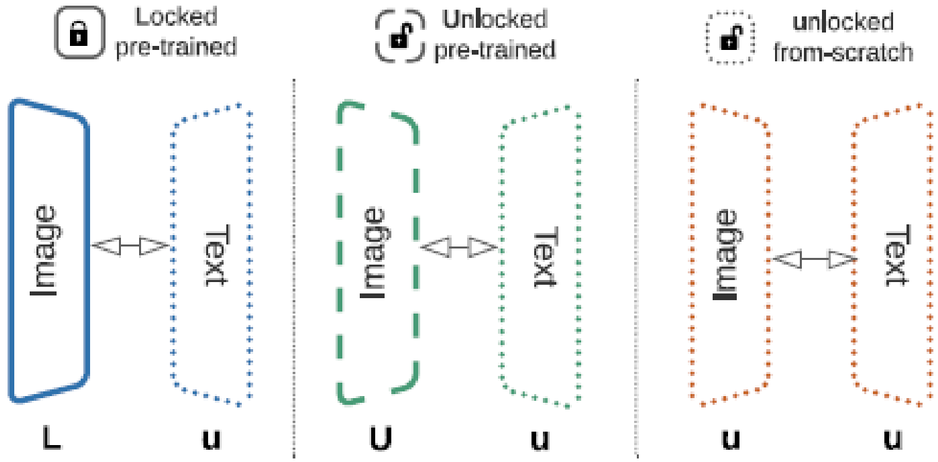
- **LiT (Locked-image Tuning)**

  - CLIP의 visual encoder가 contrastive learning 학습 과정에서 성능이 희생된다고 주장

  - 이미지 모델의 파라미터를 고정(Locked)하고, 오로지 텍스트 모델의 파라미터만을 조정하여 학습을 진행

- L : 변수는 사전 훈련된 모델에서 초기화, 학습 과정 동안 고정
- U : 변수는 사전 훈련된 모델에서 초기화, 학습 과정 동안 변경

LiT : Zero-Shot Transfer with Locked-image text Tuning



[Design choices for contrastive-tuning on image-text data]

Background

- 이미 학습된 이미지 모델 활용

  - 고품질의 (반)수동적으로 라벨링된 데이터셋을 사용해 사전 학습된 이미지 모델을 사용

  - 모델은 이미 최적의 이미지 임베딩을 생성할 수 있도록 잘 학습되어 있기 때문에, 다시 학습시킬 필요가 없음

  - 대신, 이미지를 기반으로 텍스트와의 관계만 새롭게 학습하는 것이 목표

- 대조 학습의 집중

  - 텍스트 모델이 이미지와 텍스트 간의 대조적 학습에 집중 가능

  - 즉 text encoder는 scratch하게 학습

  - image encoder가 생성한 representation을 text encoder가 읽고 해당하는 embedding을 학습

Background

- **Contrastive Learning**에서 큰 배치사이즈가 중요한 이유

  - 대량의 부정적 샘플 필요

  - Contrastive Learning의 핵심 아이디어 : 긍정적 쌍은 가까이, 부정적 쌍은 멀리

  - 배치 사이즈가 클수록 더 많은 negative sample을 학습하여 수렴

➔ CLIP의 학습 시간이 길고 자원 소모가 매우 큼

Method

methodology

- **Sigmoid loss form language image pre-training**

  - 기존의 softmax loss를 sigmoid loss로 교체

    ➔ “pair가 positive 한가 negative 한가?”

  - 이미지-텍스트 쌍을 독립적으로 처리하여, 학습 문제를 이진 분류로 전환

  - 긍정적인 쌍에 1, 부정적인 쌍에 -1 레이블 부여

```
# img_emb        : image model embedding [n, dim]
# txt_emb        : text model embedding [n, dim]
# t_prime, b     : learnable temperature and bias
# n              : mini-batch size

t = exp(t_prime)
zimg = l2_normalize(img_emb)
ztxt = l2_normalize(txt_emb)
logits = dot(zimg, ztxt.T) * t + b
labels = 2 * eye(n) - ones(n) # -1 with diagonal 1
l = -sum(log_sigmoid(labels * logits)) / n
```

부정 샘플로 인한 불균형 문제를 해결하기 위해 추가 학습 가능한 편향 b를 도입

내적을 계산하여 유사도 점수 계산

라벨 행렬 생성
2(nxn 단위 행렬) – 모든 요소 1인 nxn = 대각선 1 / 비대각선 -1

시그모이드 함수의 로그 값을 구해 손실을 계산

[sigmoid loss pseudo-implementation]

Sigmoid loss for language image pre-training

methodology

- **Sigmoid loss form language image pre-training**

  - Sigmoid loss는 이미지-텍스트 쌍을 **독립적으로 처리**

  - Softmax loss는 유사도를 평가할 때 배치 내 모든 다른 쌍들을 고려함

  - 전체 배치 작업을 요구하지 않아, softmax loss 대비 효율적이고 간단한 구현

$$-\frac{1}{2|\mathcal{B}|}\sum_{i=1}^{|\mathcal{B}|}\left(\overbrace{\log\frac{e^{t\mathbf{x}_i\cdot\mathbf{y}_i}}{\sum_{j=1}^{|\mathcal{B}|}e^{t\mathbf{x}_i\cdot\mathbf{y}_j}}}^{\text{image}\to\text{text softmax}}+\overbrace{\log\frac{e^{t\mathbf{x}_i\cdot\mathbf{y}_i}}{\sum_{j=1}^{|\mathcal{B}|}e^{t\mathbf{x}_j\cdot\mathbf{y}_i}}}^{\text{text}\to\text{image softmax}}\right)$$

[softmax loss function]

$$-\frac{1}{|\mathcal{B}|}\sum_{i=1}^{|\mathcal{B}|}\sum_{j=1}^{|\mathcal{B}|}\underbrace{\log\frac{1}{1+e^{z_{ij}(-t\mathbf{x}_i\cdot\mathbf{y}_j+b)}}}_{\mathcal{L}_{ij}}$$

[sigmoid loss function of SigLIP]

Efficient “chunked” implementation

methodology

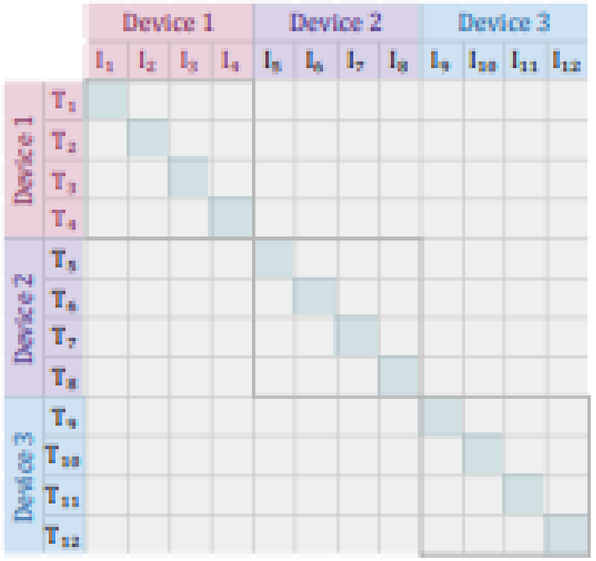
- **chunked**

  - Contrastive learning 은 데이터 병렬 처리를 활용함

  - 데이터를 D devices(여러 장치)로 분할하여 softmax loss를 계산

    ➔ 모든 임베딩을 모으는 작업**(all-gather)**

    ➔ 메모리 소모가 큰 **|B| × |B| 행렬** 만드는 작업 (모든 쌍 간의 **유사도 행렬**)



➔ sigmoid loss를 사용해 개선

- deivce에 있는 pair를 계산하고 loss를 단순합하는 과정 반복

- **|B|×b** (각 장치에서 독립적으로 처리되는 데이터의 개수)

Efficient “chunked” implementation

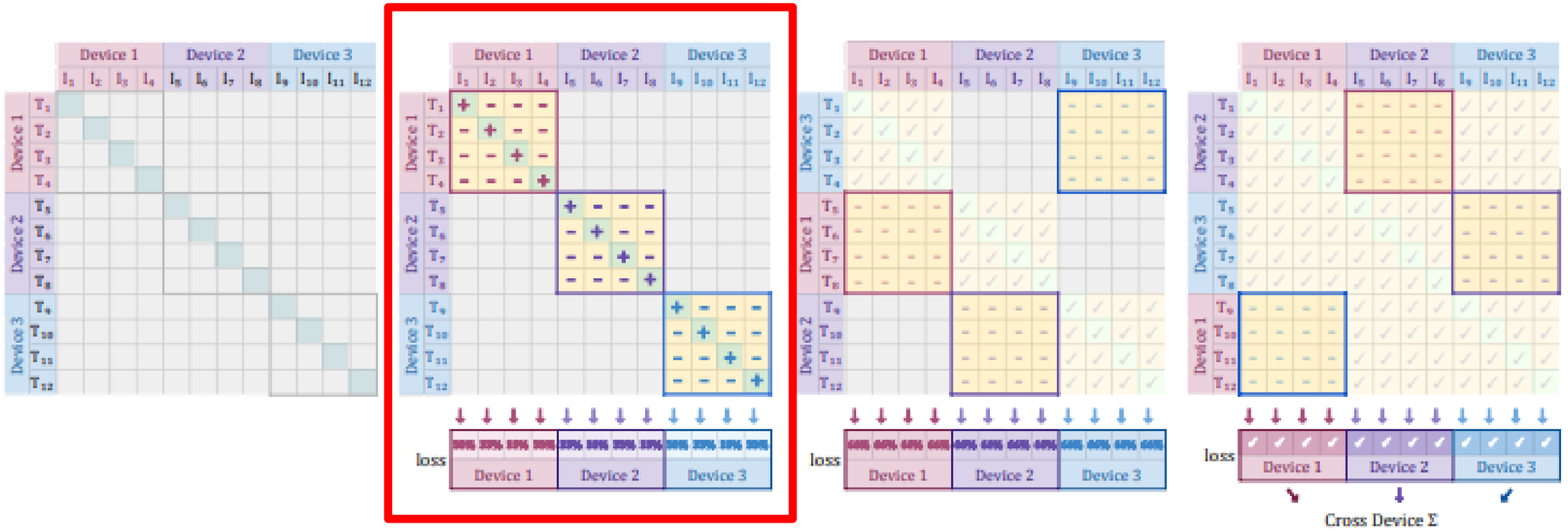
methodology

- **Chunked**

  - positive pair와 negative pair의 loss를 계산

$$-\frac{1}{|\mathcal{B}|}\underbrace{\sum_{d_i=1}^{D}}_{\text{A: } \forall \text{ device } d_i}\ \underbrace{\sum_{d_j=1}^{D}}_{\text{B: swap negs across devices}}\ \underbrace{\sum_{i=bd_i}^{b(d_i+1)}\ \sum_{j=bd_j}^{b(d_j+1)}}_{\text{C: per device loss}} \mathcal{L}_{ij}$$

$\underbrace{\phantom{\sum_{i=bd_i}^{b(d_i+1)}}}_{\text{all local positives}}$ $\underbrace{\phantom{\sum_{j=bd_j}^{b(d_j+1)}}}_{\text{negs from next device}}$



[Efficient loss implementation]

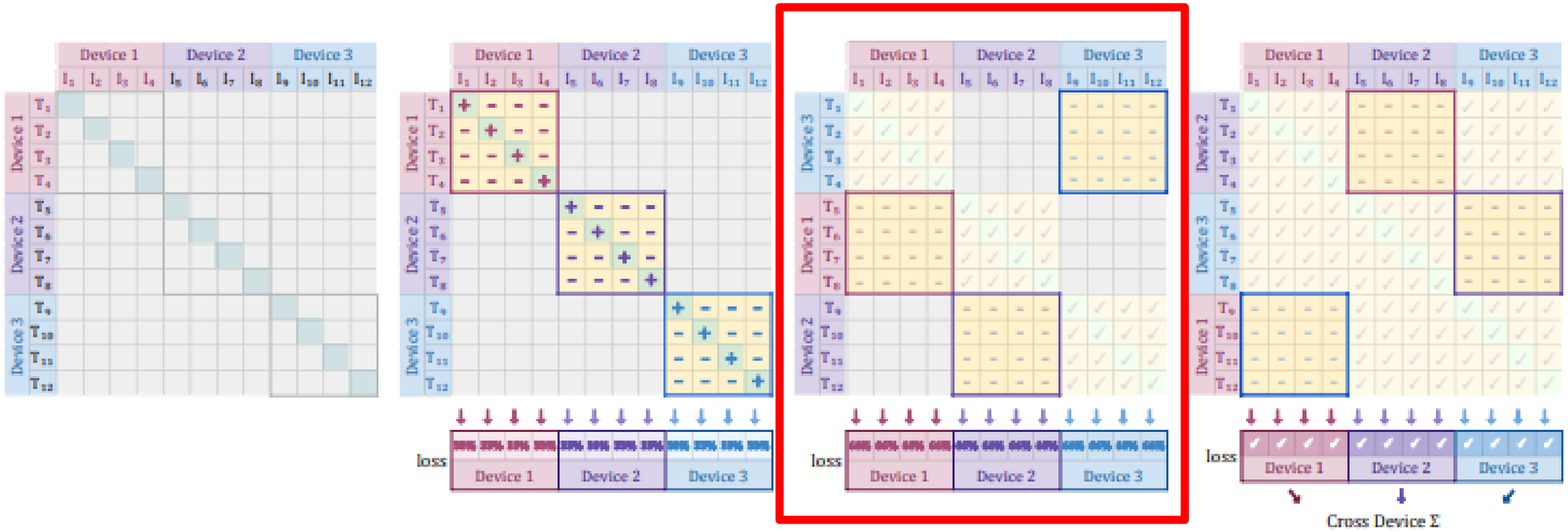
Efficient “chunked” implementation

methodology

- **Chunked**

  - 이웃 장치와 부정 샘플을 교환하여 새로운 loss를 계산

$$-\frac{1}{|\mathcal{B}|}\underbrace{\sum_{d_i=1}^{D}}_{\text{A: } \forall \text{ device } d_i}\ \underbrace{\sum_{d_j=1}^{D}}_{\text{B: swap negs across devices}}\ \underbrace{\sum_{i=bd_i}^{b(d_i+1)}\ \sum_{j=bd_j}^{b(d_j+1)}\mathcal{L}_{ij}}_{\text{C: per device loss}}$$



[Efficient loss implementation]

Efficient “chunked” implementation

methodology

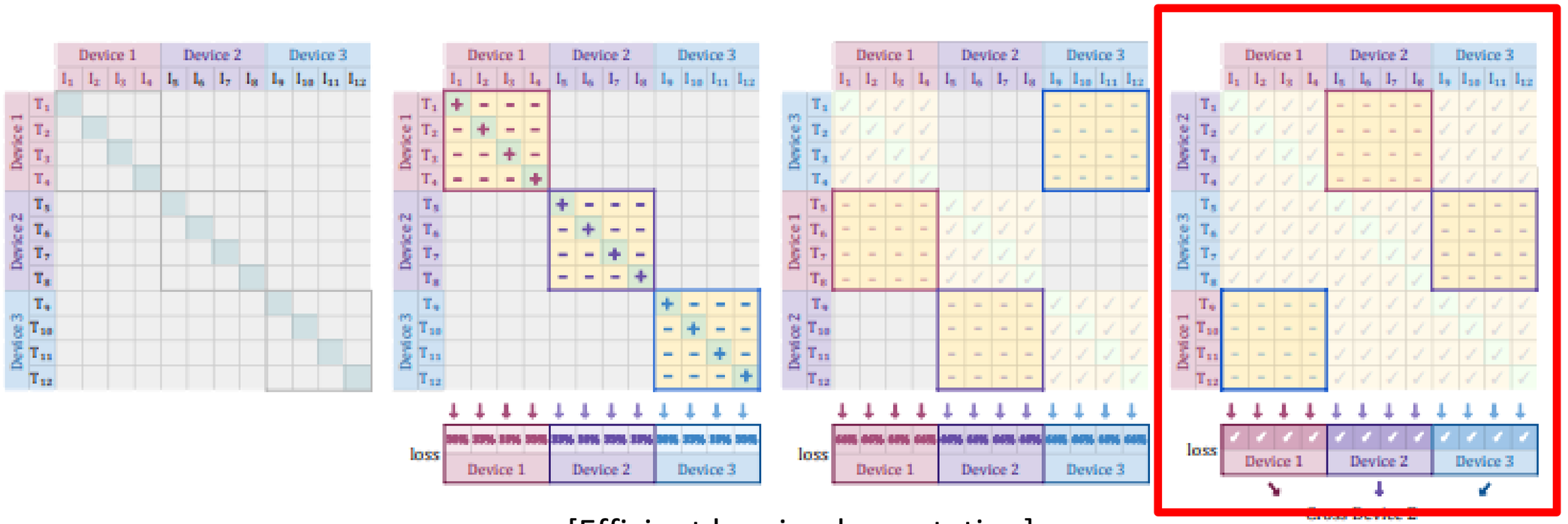
$$-\frac{1}{|B|}\underbrace{\sum_{d_i=1}^{D}}_{\text{A: } \forall \text{ device } d_i}\ \underbrace{\sum_{d_j=1}^{D}}_{\text{B: swap negs across devices}}\ \underbrace{\sum_{i=bd_i}^{b(d_i+1)}\ \sum_{j=bd_j}^{b(d_j+1)}}_{\text{C: per device loss}} \mathcal{L}_{ij}$$

$\underbrace{\sum_{i=bd_i}^{b(d_i+1)}}_{\text{all local positives}}$ $\underbrace{\sum_{j=bd_j}^{b(d_j+1)}}_{\text{negs from next device}}$

- **Chunked**

  - 모든 device 간 교환과 loss 계산이 완료되면 해당 값을 합산해 최종 loss로 사용

  - 필요한 메모리는 B^2 에서 $\frac{B^2}{n}$ 으로 감소



[Efficient loss implementation]

Efficient “chunked” implementation

methodology

- **Chunked**

- device 별 배치 사이즈를 $b = \frac{|B|}{D}$ 로 정의(b = device의 크기 / D = devic의 수 / B = 전체 배치)

- 각 device에서 로컬 손실 계산(sum C) : 긍정 쌍과 b-1개의 부정 쌍에 해당하는 loss 계산
- 부정 쌍 교환 (sum B) : 이웃 device로 부터 부정 샘플 가져와 다시 loss 계산
- 전역 손실 합산 (sum A) : 계산된 loss를 합산해 최종 loss 구함

$$-\frac{1}{|\mathcal{B}|}\underbrace{\sum_{d_i=1}^{D}}_{\text{A: } \forall \text{ device } d_i}\ \overbrace{\sum_{d_j=1}^{D}}^{\text{B: swap negs across devices}}\ \overbrace{\underbrace{\sum_{i=bd_i}^{b(d_i+1)}}_{\text{all local positives}}\ \underbrace{\sum_{j=bd_j}^{b(d_j+1)}}_{\text{negs from next device}}\mathcal{L}_{ij}}^{\text{C: per device loss}}$$

Results

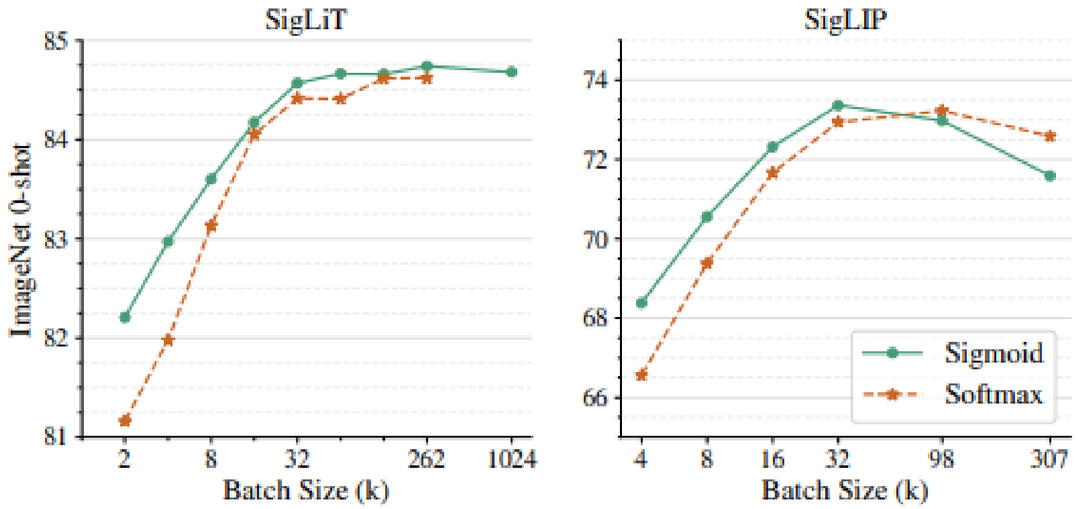
The effect of pre-training batch size

- **SigLiT 결과**

  - Sigmoid loss가 작은 배치 사이즈에서 Softmax loss를 능가하는 성능을 보임

- **SigLIP 결과**

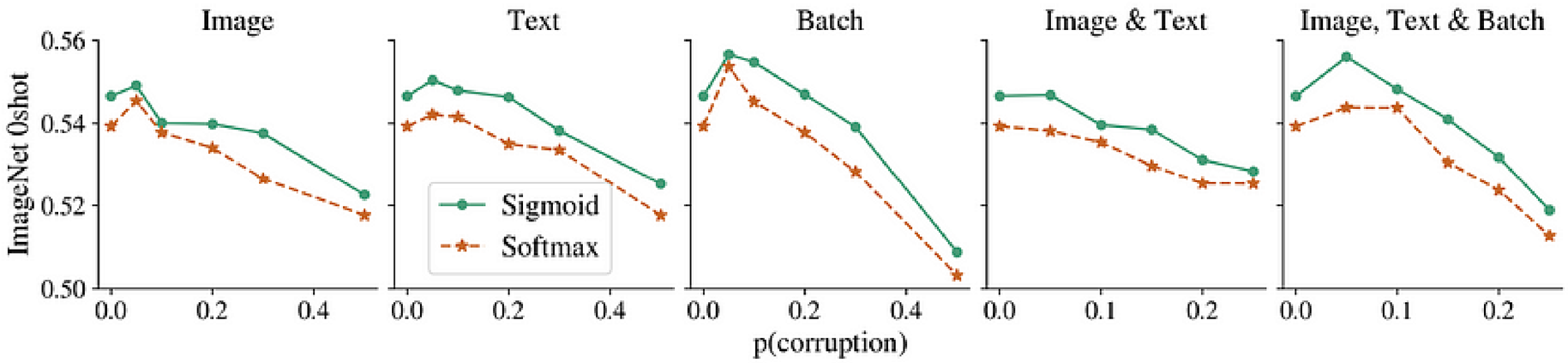
- Sigmoid loss가 더 일찍 성능 정점에 도달함
- 매우 큰 배치 사이즈에는 두 loss 모두 성능 저하를 가져옴



Results

Sigmoid-training increases robustness to data noise

- Image-Text Pair 는 강건성이 매우 중요함
- Sigmoid loss를 사용한 훈련이 Softmax loss 기반 모델보다 데이터 노이즈에 더 강건한 성능을 보임
- Corruption이 클 수록 Sigmoid loss의 성능이 더 우수함



Conclusion

- 본 연구는 Sigmoid loss를 활용한 효율적인 학습 방법을 제시함
- Sigmoid loss가 softmax loss보다 작은 훈련 배치 사이즈에서 더 좋은 성능을 보임
- Sigmoid loss는 메모리 효율성이 높아 추가 자원 없이도 더 큰 훈련 배치 사이즈를 처리할 수 있음
- 기존 방법보다 robustness를 향상시킴