Memory Bank vs DDP

- current approach uses DDP with batch size of 256
- we use two GPUs, so combined batch size is 512
- negative samples are gathered from all devices to get more negative samples [1]
- as shown in experiments with supervised teacher (SHRe), significantly improves performance
- problem is, it is expensive -> we need two GPUs instead of one
- even though batch size is doubled, training time is not halved, due to distributed communication overhead
  - comes from gradient synchronization and gathering negative samples across devices
- before we search for alternatives for more negative samples, we study the effect of using the same approach described prior, i.e. DDP with batch size of 512, but we do not gather negative samples from all devices, as done in the previous experiments
- means effective batch size is 512, but ITC done on 256 negative samples
- results will show us how important increased number of negative samples truly is for the performance of our model
- search for alternatives, that allows for large number of negative samples, but without the need for two GPUs
- one alternative is to use a FIFO queue based memory bank
- memory bank stores image and text embeddings of previous batches
- with every batch, we update the memory bank with the current batch embeddings, while discarding the oldest embeddings
- embeddings from memory bank are then used, together with the embeddings of the current batch, as negative examples
- given a batch size of N, and a memory bank size of M, we can have $N - 1 + M$ negative samples
- given a batch size of 256, and a memory bank size of 256, we can have 511 negative samples, which is the same as having an effective batch size of 512, or using two GPUs with DDP and a batch size of 256, as done in the previous experiments
- based on linear scaling rule (Equation 1) [2], [3], we reduce the learning rate from 2e-4 to 1e-4
- scaling rule is given by:

$$\texttt{lr} = \text{BaseLR} * \frac{\text{BatchSize}}{256} \tag{1}$$

- we use a batch size of 256 per device, and a base learning rate of 1e-4 ($\text{BaseLR} = 0.0001$)
- for two devices, we have an effective batch size of 512, that is why we used a learning rate of 2e-4 in the previous experiments (1e-4*512/256=2e-4)
- doing this, we observe a significant drop in performance

Model	ImageNet-1k Accuracy	Retrieval MSCOCO	Retrieval Flickr30K	Batch Size	ITC # Negative Examples	Wall Clock Time (in h)
CLIP [4]	**72.6**	**66.73**	**90.1**	32,768	32,767	-
Sx3HRe$_{\textbf{DDP}}$	26.1	66.3	42.4	512	511	7.09
Sx3HRe$_{\textbf{MB}}$	17.8	54.4	30.3	256	511	11.88
Sx3HRe$_{\textbf{DDP NG}}$	17.8	54.4	30.3	512	255	7.09

Table 1: CLIP

- might come as a surprise, but is a well known
- problem lies is rapidly changing model weights during training
- when using actual batch size of 512, or DDP, all negative examples (the representations) come from the same model with the same weights
- when using a memory bank, the negative examples are stored from previous batches, or steps respectively
- means from an older model with different weights
- leads to a fast shift in embedding space, and in the embeddings in general
- so pronounced, that even representations from the immediate previous steps are so different to the represenations of the current step, that they are not useful as negative examples
- memory bank was initially introduced as a mapping of the complete training dataset
- embedding of each sample is stored in the memory bank
- for each batch, K samples are drawn from the memory bank to be used as negative examples
- representations of samples that are currently in the batch are then updated in the memory bank
- has the same problem that representations come from an older model with different weights, especially if the dataset is large
- this is partly mitigated by using a proximal regularization, show in Equation 2

$$-\log h(i, \boldsymbol{v}_i^{t-1}) + \lambda * \|\boldsymbol{v}_i^t - \boldsymbol{v}_i^{t-1}\|_2^2 \tag{2}$$

- term $-\log h(i, \boldsymbol{v}_i^{t-1})$ can be ignored for now, and just be considered as the contrastive loss for simplicity
- more interesting is proximal regularization term $\lambda * \|\boldsymbol{v}_i^t - \boldsymbol{v}_i^{t-1}\|_2^2$
- is the mean squared error between the representation $\boldsymbol{v}_i$ of a training example i in the current batch, e.g. an image, at time step t, and the representation of the same training example at time step $t - 1$ (the last update)
- enforces that representation of a training example does not change too rapidly
- λ controls how strong this regularization is
- allowes for more stable negative examples

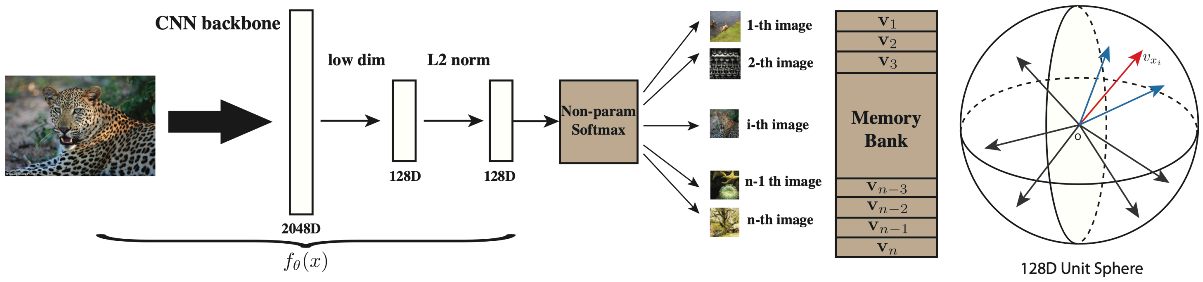

Figure 1:

- problem also identified by authors of MoCo [5]
- authors experiments, displayed in Figure 2, show that even a memory bank that maps all training examples performs significantly worse than using the actual batch size it tries to mimic
- especially with lower batch sizes
- this type of memory bank need to sample up to 65,536 ($K = 65,536$) negative examples to match actual, comparatively small, batch size of 1024

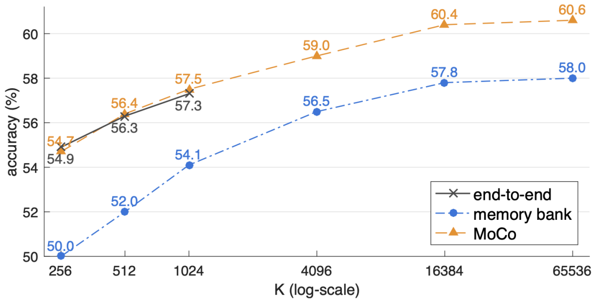

Figure 2:

- we can't use the memory bank style of [6], since we have 3,264,868 training examples
  - each embedding has a size of 768, considering storing them at full precision, so float32, we would need $3,264,868 * 768 * 4B > 10e^9 B = 10$ GB of additional GPU memory

- we suspect that using a proximal regularization term, as in Equation 2, might help to stabilize our memory bank approach

- however, we cannot use it -> the term is based on the representation of the same training example at the previous time step

- but our memory bank is significantly smaller than the training dataset, and older samples are dequeued, so the same training example will never be in the memory bank and at the same time in the current batch

- authors of MOCO propose a different approach to solving this problem

- they use a queue based memory bank, much smaller than the training dataset (cite MoCo v2), so same approach as we use

- but the negative examples in the memory bank are not updated by the model that is trained, but instead by a momentum encoder, which is a moving average of the actual model weights

- weight update of the momentum encoder is given by

$$\theta_k = m * \theta_k + (1 - m) * \theta_q \tag{3}$$

- with θ_k being the momentum encoder weights, θ_q the actual model weights, and m the momentum factor
- set to to $m = 0.999$, meaning the momentum encoder is updated very slowly
- that way negative examples in the memory bank are updated by a model with similar weights as the model that is trained
- makes them more stable and consistent
- can be seen as keeping the model consistent that produces the negative examples, instead of making the negative examples themselves consistent through an additional regularization term (Equation 2)
- disadvantage is that a copy of the same model that is being trained has to be kept in memory
- even though no gradients are needed for the momentum encoder, it still requires additional GPU memory

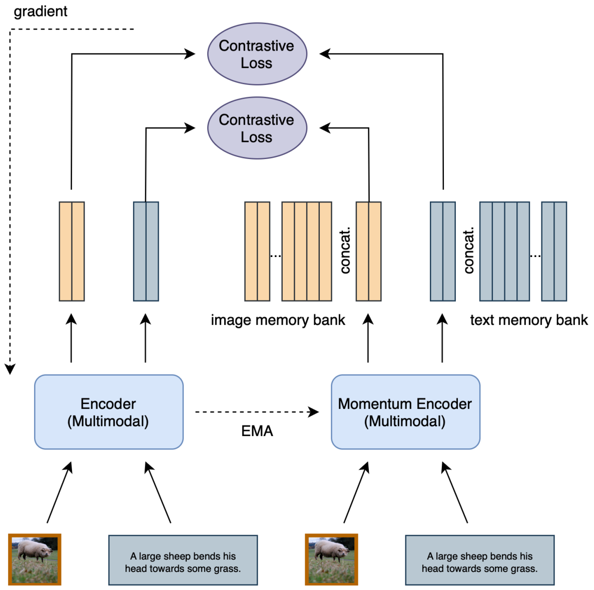
gradient
Contrastive Loss
Contrastive Loss
concat.
concat.
image memory bank
text memory bank
Encoder (Multimodal)
EMA
Momentum Encoder (Multimodal)
A large sheep bends his head towards some grass.
A large sheep bends his head towards some grass.

Figure 3: Figure inspired by MoCo v2 [7]

- in conclusion:

1. can’t use FIFO queue based memory bank, as representations of negative examples are inconsistent
2. can’t use memory bank that maps all training examples, as it would require too much additional GPU memory
3. proximal regularization not applicable to a memory bank that is smaller than the training dataset

- only alternative is to use a momentum encoder as in MoCo
- test momentum encoder -> works
- use just one gpu -> iterations per second less than with DDP, ddp with two gpus is more than twice as fast
- cost wise, two gpus are more expensive than one gpu, but training is more than twice as fast, so in the end, two gpus are cheaper
- we keep batch size of 256 per gpu, so 512 in total, and use DDP with two gpus
- combine this with ema

- because we use ddp and not deepspeed anymore, we get OOM errors
- introduce optimization: -> normal forward pass (kd+itc) -> backward and step -> free activations, gradients, and memory -> update ema -> forward pass with ema -> gather embeddings from ema over all devices -> update memory bank
- first experiments show that this approach leads much worse performance on ImageNet-1K during training
- we stop it early
- ema decay of 0.99, following MoCo v3 (MoCo v1 achives best results with 0.999) for vision Transformers
- maybe to high, try MoCo v1 with 0.999

…

- best approach would be to use no contrastive loss -> all the work we did before was because contrastive learning requires large number of negative examples
- BEiT-3 showed that this is not necessary

Bibliography

[1] H. Bao *et al.*, "VLMo: Unified Vision-Language Pre-Training with Mixture-of-Modality-Experts," in *Advances in Neural Information Processing Systems*, 2022. [Online]. Available: https://openreview.net/forum?id=bydKs84JEyw

[2] X. Chen, S. Xie, and K. He, "An Empirical Study of Training Self-Supervised Vision Transformers," *arXiv preprint arXiv:2104.02057*, 2021.

[3] P. Goyal *et al.*, "Accurate, large minibatch SGD: Training ImageNet in 1 hour," *arXiv preprint arXiv:1706.02677*, 2017.

[4] A. Radford *et al.*, "Learning transferable visual models from natural language supervision," in *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, M. Meila and T. Zhang, Eds., in Proceedings of Machine Learning Research, vol. 139. PMLR, 2021, pp. 8748–8763.

[5] K. He, H. Fan, Y. Wu, S. Xie, and R. Girshick, "Momentum Contrast for Unsupervised Visual Representation Learning," *arXiv preprint arXiv:1911.05722*, 2019.

[6] Z. Wu, Y. Xiong, S. X. Yu, and D. Lin, "Unsupervised Feature Learning via Non-parametric Instance Discrimination," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018, pp. 3733–3742. doi: 10.1109/CVPR.2018.00393.

[7] X. Chen, H. Fan, R. Girshick, and K. He, "Improved Baselines with Momentum Contrastive Learning," *arXiv preprint arXiv:2003.04297*, 2020.