Momentum Contrast for Unsupervised Visual Representation Learning(MoCo), Facebook AI Research (FAIR), 2020

`20221212. Mon.`

- 참고 논문, *“Momentum Contrast for Unsupervised Visual Representation Learning”*, IEEE, 2020.

▼ 참고

[논문 읽기] MOCO(2019), Momentum Contrast for Unsupervised Visual Representation Learning (tistory.com)

Introduction

1. contrastive loss를 사용하는 self-supervised model
2. MoCo 이전의 contrastive loss mechanism : end-to-end, memory bank 방식

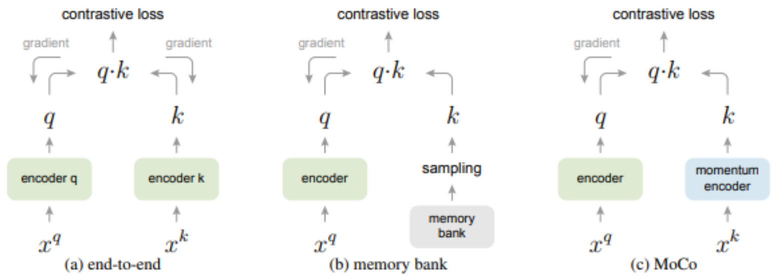

Figure 2. **Conceptual comparison of three contrastive loss mechanisms** (empirical comparisons are in Figure 3 and Table 3). Here we illustrate one pair of query and key. The three mechanisms differ in how the keys are maintained and how the key encoder is updated. **(a)**: The encoders for computing the query and key representations are updated ***end-to-end*** by back-propagation (the two encoders can be different). **(b)**: The key representations are sampled from a ***memory bank*** [61]. **(c)**: ***MoCo*** encodes the new keys on-the-fly by a momentum-updated encoder, and maintains a queue (not illustrated in this figure) of keys.

3. Contrastive loss를 최대한 활용하려면 많은 수의 negative sample가 필요하고 negative sample의 encoder는 query encoder와 일관적이어야 함
4. 이전 방식들의 단점을 개선한 것이 MoCo
    - end-to-end 방법의 문제점 : mini-batch내에 존재하는 sample들을 negative sample로 활용하는데, 많은 negative sample을 사용하려면 computational limit가 발생
    - memoty bank 방식의 문제점 : 많은 양의 negative sample을 활용할 수 있지만 encoder가 update 됨에 따라 encoded된 negetive sample은 갱신이 되지 않음

Method

- MoCo의 핵심 아이디어 : **(1) negative representation을 저장하는 queue, (2) key encoder의 mementum update**

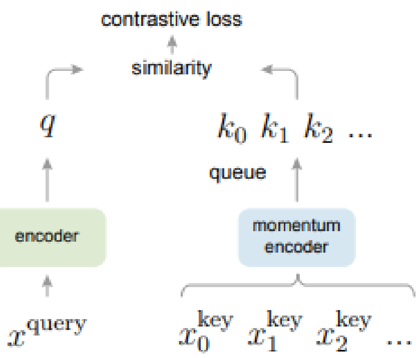

Figure 1. Momentum Contrast (MoCo) trains a visual representation encoder by matching an encoded query q to a dictionary of encoded keys using a contrastive loss. The dictionary keys $\{k_0, k_1, k_2, ...\}$ are defined on-the-fly by a set of data samples. The dictionary is built as a queue, with the current mini-batch enqueued and the oldest mini-batch dequeued, decoupling it from the mini-batch size. The keys are encoded by a slowly progressing encoder, driven by a momentum update with the query encoder. This method enables a large and consistent dictionary for learning visual representations.

  - MoCo는 하나의 이미지에 두 개의 augmentation을 적용
  - 그리고 이 이미지들은 similar로 정의
  - queue 내에 존재하는 representation은 disimiar로 사용
  - queue의 구성 : 과거 각 batch 안에서 augmentation이 적용된 image의 encoding representation들로 이루어져 있음
  - contrastive loss의 계산 : 정의된 similar와 dissimilar 사용
  - encoder를 갱신하고, decoder는 momentume update

  - 현재 batch에서 augmentation된 이미지는 queue에 enqueue
  - queue내에 존재하는 과거의 representation은 deque
    - encoder가 갱신됨에따라 과거의 representation은 consistent하지 않기 때문
  - query encoder는 학습이 진행되면서 갱신이 되고, key encoder는 momentum 기법을 사용하여 서서히 갱신

(1) Contrastive Learning as Dictionary Look-up

- InfoNCE Loss 사용

$$\mathcal{L}_q = -\log \frac{\exp(\frac{q \cdot k_+}{\tau})}{\sum_{i=0}^{K} \exp(\frac{q \cdot k_i}{\tau})}$$

(2) Momentum Contrast

- key encoder는 서서히 update

$$\theta_k \leftarrow m\theta_k + (1-m)\theta_q$$
$$where \quad m \in [0,1) \ is \ a \ momentum \ coefficient.$$

(3) Algorithm

Algorithm 1 Pseudocode of MoCo in a PyTorch-like style.

```
# f_q, f_k: encoder networks for query and key
# queue: dictionary as a queue of K keys (CxK)
# m: momentum
# t: temperature

f_k.params = f_q.params # initialize
for x in loader: # load a minibatch x with N samples
   x_q = aug(x) # a randomly augmented version
   x_k = aug(x) # another randomly augmented version

   q = f_q.forward(x_q) # queries: NxC
   k = f_k.forward(x_k) # keys: NxC
   k = k.detach() # no gradient to keys

   # positive logits: Nx1
   l_pos = bmm(q.view(N,1,C), k.view(N,C,1))

   # negative logits: NxK
   l_neg = mm(q.view(N,C), queue.view(C,K))

   # logits: Nx(1+K)
   logits = cat([l_pos, l_neg], dim=1)

   # contrastive loss, Eqn.(1)
   labels = zeros(N) # positives are the 0-th
   loss = CrossEntropyLoss(logits/t, labels)

   # SGD update: query network
   loss.backward()
   update(f_q.params)

   # momentum update: key network
   f_k.params = m*f_k.params+(1-m)*f_q.params

   # update dictionary
   enqueue(queue, k) # enqueue the current minibatch
   dequeue(queue) # dequeue the earliest minibatch
```

bmm: batch matrix multiplication; mm: matrix multiplication; cat: concatenation.