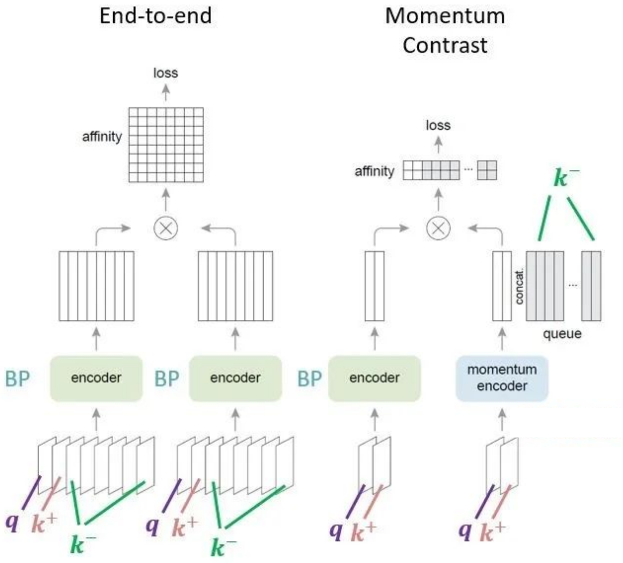
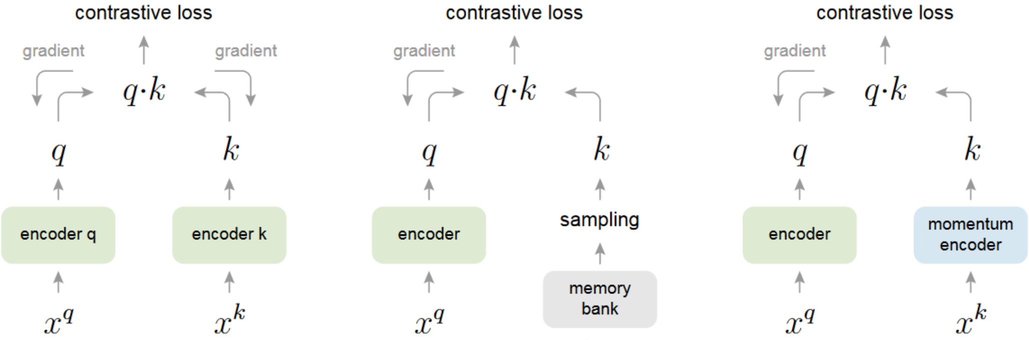
Contrastive Learning

- SimCLR
- Moco
- SimSias
- BYOL

MoCo

- End-to-end (Naïve Contrastive learning):
  - large batch size and large graphic memory
- Memory bank:
  - use a bank to store negative representations
  - clip gradient for the second branch
  - q and k may be out of synch
- MoCo
  - memory bank + momentum encoder
  - decouple batch size with number of negative samples

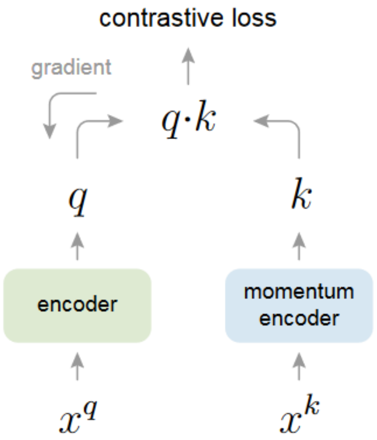




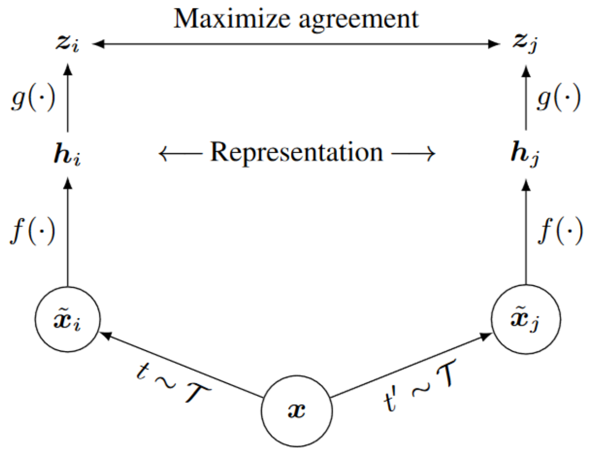
(a) end-to-end (b) memory bank (c) MoCo

SimCLR

- Consider both similarity and dis- similarity to avoid collapse
- Compared with MoCo:
  - add a projection head $g(\cdot)$, which will be discarded for downstream tasks
  - try more data augmentation techniques; use large batch size
  - use more negative terms in the loss (2N-1 vs. K)



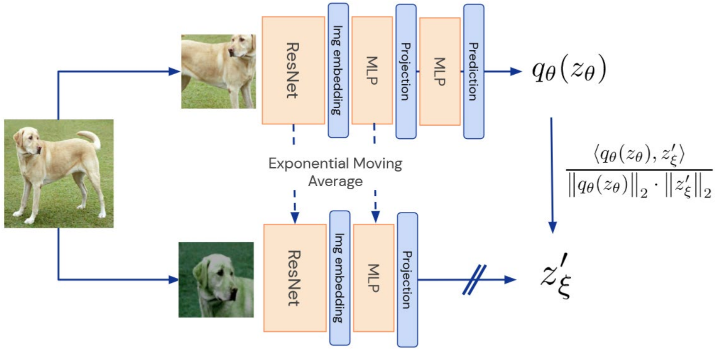
$$\mathcal{L}_q = -\log \frac{\exp(q \cdot k_+ / \tau)}{\sum_{i=0}^{K} \exp(q \cdot k_i / \tau)}$$

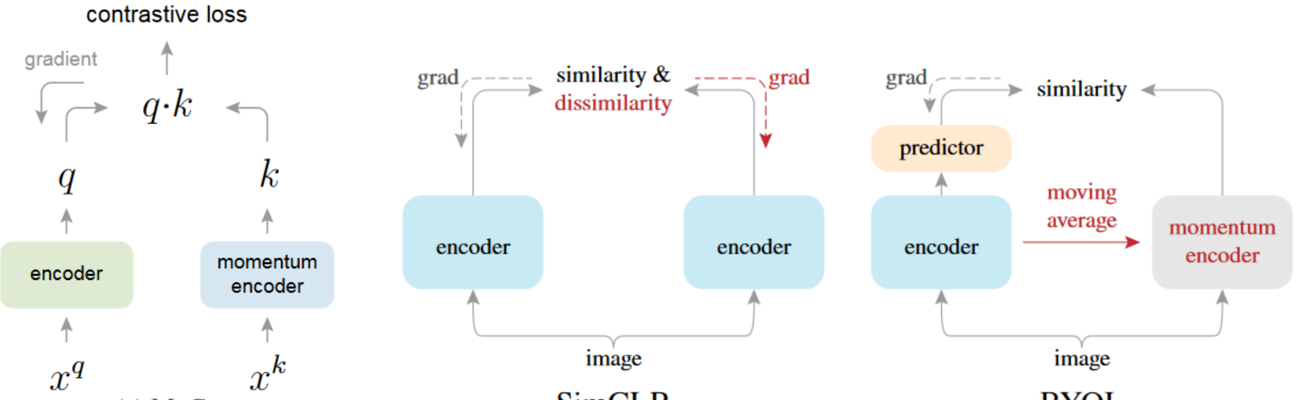


$$\ell_{i,j} = -\log \frac{\exp(\mathrm{sim}(\boldsymbol{z}_i, \boldsymbol{z}_j)/\tau)}{\sum_{k=1}^{2N} \mathbb{1}_{[k \neq i]} \exp(\mathrm{sim}(\boldsymbol{z}_i, \boldsymbol{z}_k)/\tau)}$$

BYOL

- No negative samples
  - so need approaches to prevent the network from collapsing to trivial solution
  - asymmetric structure: a predictor (MLP) to predict different views
  - gradient frozen for the second branch
  - Require global BN, thus is slow.


ResNet
Img embedding
MLP
Projection
MLP
Prediction
$q_\theta(z_\theta)$
Exponential Moving Average
$\frac{\langle q_\theta(z_\theta), z'_\xi \rangle}{\|q_\theta(z_\theta)\|_2 \cdot \|z'_\xi\|_2}$
ResNet
Img embedding
MLP
Projection
z'_ξ



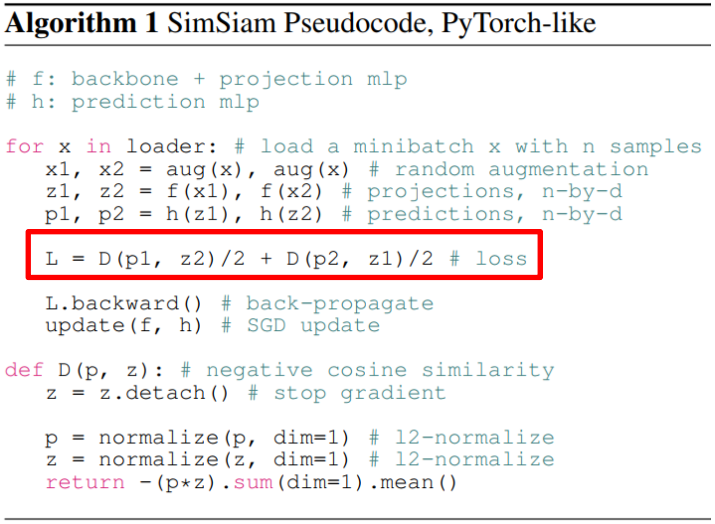
(c) MoCo

SimCLR

BYOL

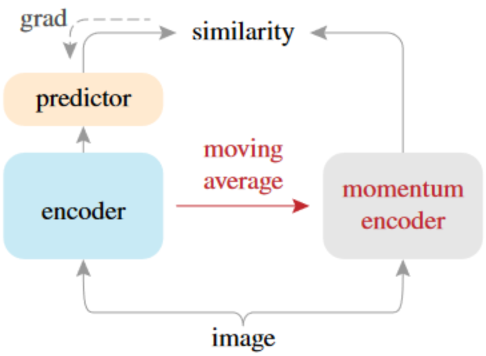
SimSiam

- SimSiam
  - Similar to BYOL, SimSiam uses the loss of two symmetrical terms
  - BYOL without momentum encoder
  - Show that the key for preventing collapsing Is
    - Stop-grad
    - Not symmetrical loss or momentum encoder

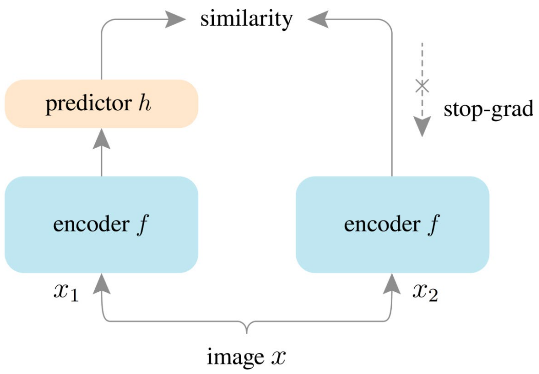


Algorithm 1 SimSiam Pseudocode, PyTorch-like

```
# f: backbone + projection mlp
# h: prediction mlp

for x in loader: # load a minibatch x with n samples
    x1, x2 = aug(x), aug(x) # random augmentation
    z1, z2 = f(x1), f(x2) # projections, n-by-d
    p1, p2 = h(z1), h(z2) # predictions, n-by-d

    L = D(p1, z2)/2 + D(p2, z1)/2 # loss

    L.backward() # back-propagate
    update(f, h) # SGD update

def D(p, z): # negative cosine similarity
    z = z.detach() # stop gradient

    p = normalize(p, dim=1) # l2-normalize
    z = normalize(z, dim=1) # l2-normalize
    return -(p*z).sum(dim=1).mean()
```



BYOL



SimSiam

Summary

- All approaches use the shared encoder in two branches
- If no negative samples, (predictor MLP + stop-gradient) is required to avoid collapsing
- Projection MLP can improve accuracy

	Momentum Encoder	Stop-Gradient for the second encoder	Predictor MLP	Negative Representations	Projection MLP (discard after training)
MoCo (v1) CVPR 20’	√	√	×	√	× (added in v2)
SimCLR ICML 20’	×	×	×	√	√
BYOL NIPS 20’	√	√	√	×	√
SimSiam CVPR 21’	×	√	√	×	√