Exploring Simple Siamese Representation Learning

Xinlei Chen Kaiming He

Facebook AI Research (FAIR)

Abstract

Siamese networks have become a common structure in various recent models for unsupervised visual representation learning. These models maximize the similarity between two augmentations of one image, subject to certain conditions for avoiding collapsing solutions. In this paper, we report surprising empirical results that ***simple Siamese*** *networks can learn meaningful representations even using* ***none*** *of the following: (i) negative sample pairs, (ii) large batches, (iii) momentum encoders. Our experiments show that collapsing solutions do exist for the loss and structure, but a stop-gradient operation plays an essential role in preventing collapsing. We provide a hypothesis on the implication of stop-gradient, and further show proof-of-concept experiments verifying it. Our "SimSiam" method achieves competitive results on ImageNet and downstream tasks. We hope this simple baseline will motivate people to rethink the roles of Siamese architectures for unsupervised representation learning. Code will be made available.*

1. Introduction

Recently there has been steady progress in un-/self-supervised representation learning, with encouraging results on multiple visual tasks (*e.g.*, [2, 17, 8, 15, 7]). Despite various original motivations, these methods generally involve certain forms of Siamese networks [4]. Siamese networks are weight-sharing neural networks applied on two or more inputs. They are natural tools for *comparing* (including but not limited to "*contrasting*") entities. Recent methods define the inputs as two augmentations of one image, and maximize the similarity subject to different conditions.

An undesired trivial solution to Siamese networks is all outputs "collapsing" to a constant. There have been several general strategies for preventing Siamese networks from collapsing. Contrastive learning [16], *e.g.*, instantiated in SimCLR [8], repulses different images (negative pairs) while attracting the same image's two views (positive pairs). The negative pairs preclude constant outputs from the solution space. Clustering [5] is another way of avoiding constant output, and SwAV [7] incorporates online clustering into Siamese networks. Beyond contrastive learning and clustering, BYOL [15] relies only on positive pairs but it does not collapse in case a momentum encoder is used.

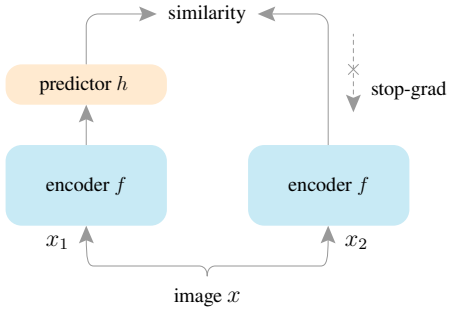


Figure 1. **SimSiam architecture**. Two augmented views of one image are processed by the same encoder network f (a backbone plus a projection MLP). Then a prediction MLP h is applied on one side, and a stop-gradient operation is applied on the other side. The model maximizes the similarity between both sides. It uses neither negative pairs nor a momentum encoder.

In this paper, we report that simple Siamese networks can work surprisingly well with *none* of the above strategies for preventing collapsing. Our model directly maximizes the similarity of one image's two views, using *neither* negative pairs *nor* a momentum encoder. It works with typical batch sizes and does not rely on large-batch training. We illustrate this "SimSiam" method in Figure 1.

Thanks to the conceptual simplicity, SimSiam can serve as a hub that relates several existing methods. In a nutshell, our method can be thought of as "BYOL *without* the momentum encoder". Unlike BYOL but like SimCLR and SwAV, our method directly shares the weights between the two branches, so it can also be thought of as "SimCLR *without* negative pairs", and "SwAV *without* online clustering". Interestingly, SimSiam is related to each method by removing one of its core components. Even so, SimSiam does not cause collapsing and can perform competitively.

We empirically show that collapsing solutions *do* exist, but a stop-gradient operation (Figure 1) is critical to prevent such solutions. The importance of stop-gradient suggests that there should be a different underlying optimization problem that is being solved. We hypothesize that there are implicitly two sets of variables, and SimSiam behaves like alternating between optimizing each set. We provide proof-of-concept experiments to verify this hypothesis.

Our simple baseline suggests that the Siamese architectures can be an essential reason for the common success of the related methods. Siamese networks can naturally introduce inductive biases for modeling invariance, as by definition "invariance" means that two observations of the same concept should produce the same outputs. Analogous to convolutions [25], which is a successful inductive bias via weight-sharing for modeling translation-invariance, the weight-sharing Siamese networks can model invariance w.r.t. more complicated transformations (*e.g.*, augmentations). We hope our exploration will motivate people to rethink the fundamental roles of Siamese architectures for unsupervised representation learning.

2. Related Work

Siamese networks. Siamese networks [4] are general models for comparing entities. Their applications include signature [4] and face [34] verification, tracking [3], one-shot learning [23], and others. In conventional use cases, the inputs to Siamese networks are from different images, and the comparability is determined by supervision.

Contrastive learning. The core idea of contrastive learning [16] is to attract the positive sample pairs and repulse the negative sample pairs. This methodology has been recently popularized for un-/self-supervised representation learning [36, 30, 20, 37, 21, 2, 35, 17, 29, 8, 9]. Simple and effective instantiations of contrastive learning have been developed using Siamese networks [37, 2, 17, 8, 9].

In practice, contrastive learning methods benefit from a large number of negative samples [36, 35, 17, 8]. These samples can be maintained in a memory bank [36]. In a Siamese network, MoCo [17] maintains a queue of negative samples and turns one branch into a momentum encoder to improve consistency of the queue. SimCLR [8] directly uses negative samples coexisting in the current batch, and it requires a large batch size to work well.

Clustering. Another category of methods for unsupervised representation learning are based on clustering [5, 6, 1, 7]. They alternate between clustering the representations and learning to predict the cluster assignment. SwAV [7] incorporates clustering into a Siamese network, by computing the assignment from one view and predicting it from another view. SwAV performs online clustering under a balanced partition constraint for each batch, which is solved by the Sinkhorn-Knopp transform [10].

While clustering-based methods do not define negative exemplars, the cluster centers can play as negative prototypes. Like contrastive learning, clustering-based methods require either a memory bank [5, 6, 1], large batches [7], or a queue [7] to provide enough samples for clustering.

BYOL. BYOL [15] directly predicts the output of one view from another view. It is a Siamese network in which one branch is a momentum encoder.[1] It is hypothesized in [15] that the momentum encoder is important for BYOL to avoid collapsing, and it reports failure results if removing the momentum encoder (0.3% accuracy, Table 5 in [15]).[2] Our empirical study challenges the *necessity* of the momentum encoder for preventing collapsing. We discover that the stop-gradient operation is critical. This discovery can be obscured with the usage of a momentum encoder, which is always accompanied with stop-gradient (as it is not updated by its parameters' gradients). While the moving-average behavior may improve accuracy with an appropriate momentum coefficient, our experiments show that it is not directly related to preventing collapsing.

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()
```

3. Method

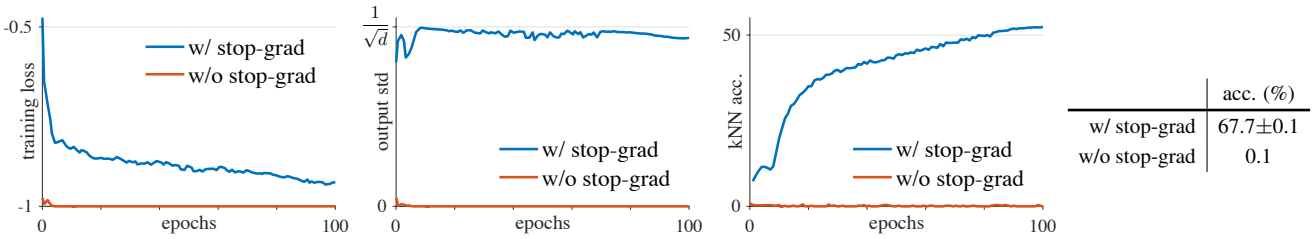
Our architecture (Figure 1) takes as input two randomly augmented views x_1 and x_2 from an image x. The two views are processed by an encoder network f consisting of a backbone (*e.g.*, ResNet [19]) and a projection MLP head [8]. The encoder f shares weights between the two views. A prediction MLP head [15], denoted as h, transforms the output of one view and matches it to the other view. Denoting the two output vectors as $p_1 \triangleq h(f(x_1))$ and $z_2 \triangleq f(x_2)$, we minimize their negative cosine similarity:

$$\mathcal{D}(p_1, z_2) = -\frac{p_1}{\|p_1\|_2} \cdot \frac{z_2}{\|z_2\|_2}, \tag{1}$$

where $\|\cdot\|_2$ is ℓ_2-norm. This is equivalent to the mean squared error of ℓ_2-normalized vectors [15], up to a scale

[1] MoCo [17] and BYOL [15] do not directly share the weights between the two branches, though in theory the momentum encoder should converge to the same status as the trainable encoder. We view these models as Siamese networks with "indirect" weight-sharing.

[2] In BYOL's arXiv v3 update, it reports 66.9% accuracy with 300-epoch pre-training when removing the momentum encoder and increasing the predictor's learning rate by 10×. Our work was done concurrently with this arXiv update. Our work studies this topic from different perspectives, with better results achieved.



	acc. (%)
w/ stop-grad	67.7±0.1
w/o stop-grad	0.1

Figure 2. **SimSiam with *vs*. without stop-gradient**. **Left plot**: training loss. Without stop-gradient it degenerates immediately. **Middle plot**: the per-channel std of the ℓ_2-normalized output, plotted as the averaged std over all channels. **Right plot**: validation accuracy of a kNN classifier [36] as a monitor of progress. **Table**: ImageNet linear evaluation ("w/ stop-grad" is mean±std over 5 trials).

of 2. Following [15], we define a symmetrized loss as:

$$\mathcal{L} = \frac{1}{2}\mathcal{D}(p_1, z_2) + \frac{1}{2}\mathcal{D}(p_2, z_1). \tag{2}$$

This is defined for each image, and the total loss is averaged over all images. Its minimum possible value is -1.

An important component for our method to work is a stop-gradient (`stopgrad`) operation (Figure 1). We implement it by modifying (1) as:

$$\mathcal{D}(p_1, \texttt{stopgrad}(z_2)). \tag{3}$$

This means that z_2 is treated as a constant in this term. Similarly, the form in (2) is implemented as:

$$\mathcal{L} = \frac{1}{2}\mathcal{D}(p_1, \texttt{stopgrad}(z_2)) + \frac{1}{2}\mathcal{D}(p_2, \texttt{stopgrad}(z_1)). \tag{4}$$

Here the encoder on x_2 receives no gradient from z_2 in the first term, but it receives gradients from p_2 in the second term (and vice versa for x_1).

The pseudo-code of SimSiam is in Algorithm 1.

Baseline settings. Unless specified, our explorations use the following settings for unsupervised pre-training:

- *Optimizer*. We use SGD for pre-training. Our method does *not* require a large-batch optimizer such as LARS [38] (unlike [8, 15, 7]). We use a learning rate of $lr \times \text{BatchSize}/256$ (linear scaling [14]), with a base $lr = 0.05$. The learning rate has a cosine decay schedule [27, 8]. The weight decay is 0.0001 and the SGD momentum is 0.9.

  The batch size is 512 by default, which is friendly to typical 8-GPU implementations. Other batch sizes also work well (Sec. 4.3). We use batch normalization (BN) [22] synchronized across devices, following [8, 15, 7].

- *Projection MLP*. The projection MLP (in f) has BN applied to each fully-connected (fc) layer, including its output fc. Its output fc has no ReLU. The hidden fc is 2048-d. This MLP has 3 layers.

- *Prediction MLP*. The prediction MLP (h) has BN applied to its hidden fc layers. Its output fc does not have BN (ablation in Sec. 4.4) or ReLU. This MLP has 2 layers. The dimension of h's input and output (z and p) is $d = 2048$, and h's hidden layer's dimension is 512, making h a bottleneck structure (ablation in supplement).

We use ResNet-50 [19] as the default backbone. Other implementation details are in supplement. We perform 100-epoch pre-training in ablation experiments.

Experimental setup. We do unsupervised pre-training on the 1000-class ImageNet training set [11] without using labels. The quality of the pre-trained representations is evaluated by training a supervised linear classifier on frozen representations in the training set, and then testing it in the validation set, which is a common protocol. The implementation details of linear classification are in supplement.

4. Empirical Study

In this section we empirically study the SimSiam behaviors. We pay special attention to what may contribute to the model's non-collapsing solutions.

4.1. Stop-gradient

Figure 2 presents a comparison on "with *vs*. without stop-gradient". The architectures and all hyper-parameters are kept unchanged, and stop-gradient is the only difference.

Figure 2 (left) shows the training loss. *Without* stop-gradient, the optimizer quickly finds a degenerated solution and reaches the minimum possible loss of -1. To show that the degeneration is caused by collapsing, we study the standard deviation (std) of the ℓ_2-normalized output $z/\|z\|_2$. If the outputs collapse to a constant vector, their std over all samples should be zero for each channel. This can be observed from the red curve in Figure 2 (middle).

As a comparison, if the output z has a zero-mean isotropic Gaussian distribution, we can show that the std of $z/\|z\|_2$ is $\frac{1}{\sqrt{d}}$.[3] The blue curve in Figure 2 (middle) shows

[3]Here is an informal derivation: denote $z/\|z\|_2$ as z', that is, $z'_i = z_i / (\sum_{j=1}^{d} z_j^2)^{\frac{1}{2}}$ for the i-th channel. If z_j is subject to an i.i.d Gaussian distribution: $z_j \sim \mathcal{N}(0, 1), \forall j$, then $z'_i \approx z_i / d^{\frac{1}{2}}$ and $\text{std}[z'_i] \approx 1/d^{\frac{1}{2}}$.

	pred. MLP h	acc. (%)
baseline	lr with cosine decay	67.7
(a)	no pred. MLP	0.1
(b)	fixed random init.	1.5
(c)	lr not decayed	68.1

Table 1. **Effect of prediction MLP** (ImageNet linear evaluation accuracy with 100-epoch pre-training). In all these variants, we use the same schedule for the encoder f (lr with cosine decay).

that with stop-gradient, the std value is near $\frac{1}{\sqrt{d}}$. This indicates that the outputs do not collapse, and they are scattered on the unit hypersphere.

Figure 2 (right) plots the validation accuracy of a k-nearest-neighbor (kNN) classifier [36]. This kNN classifier can serve as a monitor of the progress. With stop-gradient, the kNN monitor shows a steadily improving accuracy.

The linear evaluation result is in the table in Figure 2. SimSiam achieves a nontrivial accuracy of 67.7%. This result is reasonably stable as shown by the std of 5 trials. Solely removing stop-gradient, the accuracy becomes 0.1%, which is the chance-level guess in ImageNet.

Discussion. Our experiments show that *there exist collapsing solutions*. The collapse can be observed by the minimum possible loss and the constant outputs.[4] The existence of the collapsing solutions implies that it is *insufficient* for our method to prevent collapsing *solely* by the architecture designs (*e.g.*, predictor, BN, ℓ_2-norm). In our comparison, all these architecture designs are kept unchanged, but they do not prevent collapsing if stop-gradient is removed.

The introduction of stop-gradient implies that there should be another optimization problem that is being solved underlying. We propose a hypothesis in Sec. 5.

4.2. Predictor

In Table 1 we study the predictor MLP's effect.

The model does not work if removing h (Table 1a), *i.e.*, h is the identity mapping. Actually, this observation can be expected if the symmetric loss (4) is used. Now the loss is $\frac{1}{2}\mathcal{D}(z_1, \texttt{stopgrad}(z_2)) + \frac{1}{2}\mathcal{D}(z_2, \texttt{stopgrad}(z_1))$. Its gradient has the same direction as the gradient of $\mathcal{D}(z_1, z_2)$, with the magnitude scaled by $1/2$. In this case, using stop-gradient is equivalent to removing stop-gradient and scaling the loss by 1/2. Collapsing is observed (Table 1a).

We note that this derivation on the gradient direction is valid only for the symmetrized loss. But we have observed that the *asymmetric* variant (3) also fails if removing h, while it can work if h is kept (Sec. 4.6). These experiments suggest that h is helpful for our model.

If h is fixed as random initialization, our model does not work either (Table 1b). However, this failure is *not* about collapsing. The training does not converge, and the loss remains high. The predictor h should be trained to adapt to the representations.

We also find that h with a constant lr (without decay) can work well and produce even better results than the baseline (Table 1c). A possible explanation is that h should adapt to the latest representations, so it is not necessary to force it converge (by reducing lr) before the representations are sufficiently trained. In many variants of our model, we have observed that h with a constant lr provides slightly better results. We use this form in the following subsections.

[4] We note that a chance-level accuracy (0.1%) is not sufficient to indicate collapsing. A model with a diverging loss, which is another pattern of failure, may also exhibit a chance-level accuracy.

batch size	64	128	256	512	1024	2048	4096
acc. (%)	66.1	67.3	68.1	68.1	68.0	67.9	64.0

Table 2. **Effect of batch sizes** (ImageNet linear evaluation accuracy with 100-epoch pre-training).

	case	proj. MLP's BN hidden	proj. MLP's BN output	pred. MLP's BN hidden	pred. MLP's BN output	acc. (%)
(a)	none	-	-	-	-	34.6
(b)	hidden-only	✓	-	✓	-	67.4
(c)	default	✓	✓	✓	-	68.1
(d)	all	✓	✓	✓	✓	unstable

Table 3. **Effect of batch normalization on MLP heads** (ImageNet linear evaluation accuracy with 100-epoch pre-training).

4.3. Batch Size

Table 2 reports the results with a batch size from 64 to 4096. When the batch size changes, we use the same linear scaling rule ($lr \times$BatchSize/256) [14] with base $lr = 0.05$. We use 10 epochs of warm-up [14] for batch sizes ≥ 1024. Note that we keep using the same SGD optimizer (rather than LARS [38]) for all batch sizes studied.

Our method works reasonably well over this wide range of batch sizes. Even a batch size of 128 or 64 performs decently, with a drop of 0.8% or 2.0% in accuracy. The results are similarly good when the batch size is from 256 to 2048, and the differences are at the level of random variations.

This behavior of SimSiam is noticeably different from SimCLR [8] and SwAV [7]. All three methods are Siamese networks with direct weight-sharing, but SimCLR and SwAV both require a large batch (*e.g.*, 4096) to work well.

We also note that the standard SGD optimizer does not work well when the batch is too large (even in supervised learning [14, 38]), and our result is lower with a 4096 batch. We expect a specialized optimizer (*e.g.*, LARS [38]) will help in this case. However, our results show that a specialized optimizer is *not* necessary for preventing collapsing.

4.4. Batch Normalization

Table 3 compares the configurations of BN on the MLP heads. In Table 3a we remove all BN layers in the MLP heads (10-epoch warmup [14] is used specifically for this

entry). This variant does *not* cause collapse, although the accuracy is low (34.6%). The low accuracy is likely because of optimization difficulty. Adding BN to the hidden layers (Table 3b) increases accuracy to 67.4%.

Further adding BN to the output of the *projection* MLP (*i.e.*, the output of f) boosts accuracy to 68.1% (Table 3c), which is our default configuration. In this entry, we also find that the learnable affine transformation (scale and offset [22]) in f's output BN is not necessary, and disabling it leads to a comparable accuracy of 68.2%.

Adding BN to the output of the prediction MLP h does not work well (Table 3d). We find that this is not about collapsing. The training is unstable and the loss oscillates.

In summary, we observe that BN is helpful for optimization when used appropriately, which is similar to BN's behavior in other *supervised* learning scenarios. But we have seen no evidence that BN helps to prevent collapsing: actually, the comparison in Sec. 4.1 (Figure 2) has exactly the same BN configuration for both entries, but the model collapses if stop-gradient is not used.

4.5. Similarity Function

Besides the cosine similarity function (1), our method also works with cross-entropy similarity. We modify $\mathcal{D}$ as: $\mathcal{D}(p_1, z_2) = -\texttt{softmax}(z_2) \cdot \log \texttt{softmax}(p_1)$. Here the softmax function is along the channel dimension. The output of softmax can be thought of as the probabilities of belonging to each of d pseudo-categories.

We simply replace the cosine similarity with the cross-entropy similarity, and symmetrize it using (4). All hyper-parameters and architectures are unchanged, though they may be suboptimal for this variant. Here is the comparison:

	cosine	cross-entropy
acc. (%)	68.1	63.2

The cross-entropy variant can converge to a reasonable result without collapsing. This suggests that the collapsing prevention behavior is not just about the cosine similarity.

This variant helps to set up a connection to SwAV [7], which we discuss in Sec. 6.2.

4.6. Symmetrization

Thus far our experiments have been based on the symmetrized loss (4). We observe that SimSiam's behavior of preventing collapsing does not depend on symmetrization. We compare with the *asymmetric* variant (3) as follows:

	sym.	asym.	asym. 2×
acc. (%)	68.1	64.8	67.3

The asymmetric variant achieves reasonable results. Symmetrization is helpful for boosting accuracy, but it is not related to collapse prevention. Symmetrization makes one more prediction for each image, and we may roughly compensate for this by sampling two pairs for each image in the asymmetric version ("2×"). It makes the gap smaller.

4.7. Summary

We have empirically shown that in a variety of settings, SimSiam can produce meaningful results without collapsing. The optimizer (batch size), batch normalization, similarity function, and symmetrization may affect accuracy, but we have seen *no* evidence that they are related to collapse prevention. It is mainly the stop-gradient operation that plays an essential role.

5. Hypothesis

We discuss a hypothesis on what is implicitly optimized by SimSiam, with proof-of-concept experiments provided.

5.1. Formulation

Our hypothesis is that SimSiam is an implementation of an Expectation-Maximization (EM) like algorithm. It implicitly involves two sets of variables, and solves two underlying sub-problems. The presence of stop-gradient is the consequence of introducing the extra set of variables.

We consider a loss function of the following form:

$$\mathcal{L}(\theta, \eta) = \mathbb{E}_{x,\mathcal{T}} \Big[\big\| \mathcal{F}_\theta(\mathcal{T}(x)) - \eta_x \big\|_2^2 \Big]. \tag{5}$$

$\mathcal{F}$ is a network parameterized by θ. $\mathcal{T}$ is the augmentation. x is an image. The expectation $\mathbb{E}[\cdot]$ is over the distribution of images and augmentations. For the ease of analysis, here we use the mean squared error $\|\cdot\|_2^2$, which is equivalent to the cosine similarity if the vectors are ℓ_2-normalized. We do not consider the predictor yet and will discuss it later.

In (5), we have introduced *another set of variables* which we denote as η. The size of η is proportional to the number of images. Intuitively, η_x is the representation of the image x, and the subscript $_x$ means using the image index to access a sub-vector of η. η is *not necessarily* the output of a network; it is the argument of an optimization problem.

With this formulation, we consider solving:

$$\min_{\theta,\eta} \mathcal{L}(\theta, \eta). \tag{6}$$

Here the problem is w.r.t. both θ and η. This formulation is analogous to k-means clustering [28]. The variable θ is analogous to the clustering centers: it is the learnable parameters of an encoder. The variable η_x is analogous to the assignment vector of the sample x (a one-hot vector in k-means): it is the representation of x.

Also analogous to k-means, the problem in (6) can be solved by an alternating algorithm, fixing one set of variables and solving for the other set. Formally, we can alternate between solving these two subproblems:

$$\theta^t \leftarrow \arg\min_\theta \mathcal{L}(\theta, \eta^{t-1}) \tag{7}$$

$$\eta^t \leftarrow \arg\min_\eta \mathcal{L}(\theta^t, \eta) \tag{8}$$

Here t is the index of alternation and "$\leftarrow$" means assigning.

Solving for θ. One can use SGD to solve the sub-problem (7). *The stop-gradient operation is a natural consequence*, because the gradient does not back-propagate to η^{t-1} which is a constant in this subproblem.

Solving for η. The sub-problem (8) can be solved independently for each η_x. Now the problem is to minimize: $\mathbb{E}_{\mathcal{T}}\Big[\|\mathcal{F}_{\theta^t}(\mathcal{T}(x)) - \eta_x\|_2^2\Big]$ for each image x, noting that the expectation is over the distribution of augmentation $\mathcal{T}$. Due to the mean squared error,[5] it is easy to solve it by:

$$\eta_x^t \leftarrow \mathbb{E}_{\mathcal{T}}\Big[\mathcal{F}_{\theta^t}(\mathcal{T}(x))\Big]. \tag{9}$$

This indicates that η_x is assigned with the *average* representation of x over the distribution of augmentation.

One-step alternation. SimSiam can be approximated by one-step alternation between (7) and (8). First, we approximate (9) by sampling the augmentation only *once*, denoted as $\mathcal{T}'$, and ignoring $\mathbb{E}_{\mathcal{T}}[\cdot]$:

$$\eta_x^t \leftarrow \mathcal{F}_{\theta^t}(\mathcal{T}'(x)). \tag{10}$$

Inserting it into the sub-problem (7), we have:

$$\theta^{t+1} \leftarrow \arg\min_{\theta} \mathbb{E}_{x,\mathcal{T}}\Big[\big\|\mathcal{F}_{\theta}(\mathcal{T}(x)) - \mathcal{F}_{\theta^t}(\mathcal{T}'(x))\big\|_2^2\Big]. \tag{11}$$

Now θ^t is a constant in this sub-problem, and $\mathcal{T}'$ implies another view due to its random nature. This formulation exhibits the Siamese architecture. Second, if we implement (11) by *reducing* the loss with one SGD step, then we can approach the SimSiam algorithm: a Siamese network naturally with stop-gradient applied.

Predictor. Our above analysis does not involve the predictor h. We further assume that h is helpful in our method because of the approximation due to (10).

By definition, the predictor h is expected to minimize: $\mathbb{E}_z\Big[\big\|h(z_1) - z_2\big\|_2^2\Big]$. The optimal solution to h should satisfy: $h(z_1) = \mathbb{E}_z[z_2] = \mathbb{E}_{\mathcal{T}}\big[f(\mathcal{T}(x))\big]$ for any image x. This term is similar to the one in (9). In our approximation in (10), the expectation $\mathbb{E}_{\mathcal{T}}[\cdot]$ is ignored. The usage of h may fill this gap. In practice, it would be unrealistic to actually compute the expectation $\mathbb{E}_{\mathcal{T}}$. But it may be possible for a neural network (*e.g.*, the preditor h) to learn to predict the expectation, while the sampling of $\mathcal{T}$ is implicitly distributed across multiple epochs.

[5] If we use the cosine similarity, we can approximately solve it by ℓ_2-normalizing $\mathcal{F}$'s output and η_x.

Symmetrization. Our hypothesis does not involve symmetrization. Symmetrization is like denser sampling $\mathcal{T}$ in (11). Actually, the SGD optimizer computes the empirical expectation of $\mathbb{E}_{x,\mathcal{T}}[\cdot]$ by sampling a batch of images and one pair of augmentations ($\mathcal{T}_1$, $\mathcal{T}_2$). In principle, the empirical expectation should be more precise with denser sampling. Symmetrization supplies an extra pair ($\mathcal{T}_2$, $\mathcal{T}_1$). This explains that symmetrization is not necessary for our method to work, yet it is able to improve accuracy, as we have observed in Sec. 4.6.

5.2. Proof of concept

We design a series of proof-of-concept experiments that stem from our hypothesis. They are methods different with SimSiam, and they are designed to verify our hypothesis.

Multi-step alternation. We have hypothesized that the SimSiam algorithm is like alternating between (7) and (8), with an interval of one step of SGD update. Under this hypothesis, it is likely for our formulation to work if the interval has multiple steps of SGD.

In this variant, we treat t in (7) and (8) as the index of an outer loop; and the sub-problem in (7) is updated by an inner loop of k SGD steps. In each alternation, we pre-compute the η_x required for all k SGD steps using (10) and cache them in memory. Then we perform k SGD steps to update θ. We use the same architecture and hyper-parameters as SimSiam. The comparison is as follows:

	1-step	10-step	100-step	1-epoch
acc. (%)	68.1	68.7	68.9	67.0

Here, "1-step" is equivalent to SimSiam, and "1-epoch" denotes the k steps required for one epoch. All multi-step variants work well. The 10-/100-step variants even achieve *better* results than SimSiam, though at the cost of extra pre-computation. This experiment suggests that the alternating optimization is a valid formulation, and SimSiam is a special case of it.

Expectation over augmentations. The usage of the predictor h is presumably because the expectation $\mathbb{E}_{\mathcal{T}}[\cdot]$ in (9) is ignored. We consider another way to approximate this expectation, in which we find h is not needed.

In this variant, we do not update η_x directly by the assignment (10); instead, we maintain a moving-average: $\eta_x^t \leftarrow m * \eta_x^{t-1} + (1-m) * \mathcal{F}_{\theta^t}(\mathcal{T}'(x))$, where m is a momentum coefficient (0.8 here). This computation is similar to maintaining the *memory bank* as in [36]. This moving-average provides an approximated expectation of multiple views. This variant has 55.0% accuracy *without* the predictor h. As a comparison, it fails completely if we remove h but do not maintain the moving average (as shown in Table 1a). This proof-of-concept experiment supports that the usage of predictor h is related to approximating $\mathbb{E}_{\mathcal{T}}[\cdot]$.

method	batch size	negative pairs	momentum encoder	100 ep	200 ep	400 ep	800 ep
SimCLR (repro.+)	4096	✓		66.5	68.3	69.8	70.4
MoCo v2 (repro.+)	**256**	✓	✓	67.4	69.9	71.0	72.2
BYOL (repro.)	4096		✓	66.5	**70.6**	**73.2**	**74.3**
SwAV (repro.+)	4096			66.5	69.1	70.7	71.8
SimSiam	**256**			**68.1**	70.0	70.8	71.3

Table 4. **Comparisons on ImageNet linear classification**. All are based on **ResNet-50** pre-trained with **two 224×224 views**. Evaluation is on a single crop. All competitors are from our reproduction, and "+" denotes *improved* reproduction *vs*. original papers (see supplement).

	VOC 07 detection			VOC 07+12 detection			COCO detection			COCO instance seg.		
pre-train	AP_{50}	AP	AP_{75}	AP_{50}	AP	AP_{75}	AP_{50}	AP	AP_{75}	AP^{mask}_{50}	AP^{mask}	AP^{mask}_{75}
scratch	35.9	16.8	13.0	60.2	33.8	33.1	44.0	26.4	27.8	46.9	29.3	30.8
ImageNet supervised	74.4	42.4	42.7	81.3	53.5	58.8	58.2	38.2	41.2	54.7	33.3	35.2
SimCLR (repro.+)	75.9	46.8	50.1	81.8	55.5	61.4	57.7	37.9	40.9	54.6	33.3	35.3
MoCo v2 (repro.+)	**77.1**	**48.5**	**52.5**	**82.3**	**57.0**	**63.3**	**58.8**	**39.2**	**42.5**	**55.5**	**34.3**	**36.6**
BYOL (repro.)	**77.1**	47.0	49.9	81.4	55.3	61.1	57.8	37.9	40.9	54.3	33.2	35.0
SwAV (repro.+)	75.5	46.5	49.6	81.5	55.4	61.4	57.6	37.6	40.3	54.2	33.1	35.1
SimSiam, base	75.5	47.0	50.2	**82.0**	56.4	62.8	57.5	37.9	40.9	54.2	33.2	35.2
SimSiam, optimal	**77.3**	**48.5**	**52.5**	**82.4**	**57.0**	**63.7**	**59.3**	**39.2**	**42.1**	**56.0**	**34.4**	**36.7**

Table 5. **Transfer Learning**. All unsupervised methods are based on 200-epoch pre-training in ImageNet. *VOC 07 detection*: Faster R-CNN [32] fine-tuned in VOC 2007 trainval, evaluated in VOC 2007 test; *VOC 07+12 detection*: Faster R-CNN fine-tuned in VOC 2007 trainval + 2012 train, evaluated in VOC 2007 test; *COCO detection* and *COCO instance segmentation*: Mask R-CNN [18] (1× schedule) fine-tuned in COCO 2017 train, evaluated in COCO 2017 val. All Faster/Mask R-CNN models are with the C4-backbone [13]. All VOC results are the average over 5 trials. **Bold entries** are within 0.5 below the best.

5.3. Discussion

Our hypothesis is about what the optimization problem can be. It does not explain why collapsing is prevented. We point out that SimSiam and its variants' non-collapsing behavior still remains as an empirical observation.

Here we briefly discuss our understanding on this open question. The alternating optimization provides a different trajectory, and the trajectory depends on the initialization. It is *unlikely* that the initialized η, which is the output of a randomly initialized network, would be a constant. Starting from this initialization, it may be difficult for the alternating optimizer to approach a constant η_x for all x, because the method does *not* compute the gradients w.r.t. η jointly for all x. The optimizer seeks another trajectory (Figure 2 left), in which the outputs are scattered (Figure 2 middle).

6. Comparisons

6.1. Result Comparisons

ImageNet. We compare with the state-of-the-art frameworks in Table 4 on ImageNet linear evaluation. For fair comparisons, all competitors are based on our reproduction, and "+" denotes *improved* reproduction *vs*. the original papers (see supplement). For each individual method, we follow the hyper-parameter and augmentation recipes in its original paper.[6] All entries are based on a standard ResNet-50, with two 224×224 views used during pre-training.

[6]In our BYOL reproduction, the 100, 200(400), 800-epoch recipes follow the 100, 300, 1000-epoch recipes in [15]: lr is {0.45, 0.3, 0.2}, wd is {1e-6, 1e-6, 1.5e-6}, and momentum coefficient is {0.99, 0.99, 0.996}.

Table 4 shows the results and the main properties of the methods. SimSiam is trained with a batch size of 256, using neither negative samples nor a momentum encoder. Despite it simplicity, SimSiam achieves competitive results. It has the highest accuracy among all methods under 100-epoch pre-training, though its gain of training longer is smaller. It has better results than SimCLR in all cases.

Transfer Learning. In Table 5 we compare the representation quality by transferring them to other tasks, including VOC [12] object detection and COCO [26] object detection and instance segmentation. We fine-tune the pre-trained models end-to-end in the target datasets. We use the public codebase from MoCo [17] for all entries, and search the fine-tuning learning rate for each individual method. All methods are based on 200-epoch pre-training in ImageNet using our reproduction.

Table 5 shows that SimSiam's representations are *transferable* beyond the ImageNet task. It is competitive among these leading methods. The "base" SimSiam in Table 5 uses the baseline pre-training recipe as in our ImageNet experiments. We find that another recipe of $lr = 0.5$ and $wd = $ 1e-5 (with similar ImageNet accuracy) can produce better results in all tasks (Table 5, "SimSiam, optimal").

We emphasize that *all these methods are highly successful for transfer learning*—in Table 5, they can surpass or be on par with the ImageNet *supervised* pre-training counterparts in all tasks. Despite many design differences, a common structure of these methods is the *Siamese network*. This comparison suggests that the Siamese structure is a core factor for their general success.

6.2. Methodology Comparisons

Beyond accuracy, we also compare the methodologies of these Siamese architectures. Our method plays as a hub to connect these methods. Figure 3 abstracts these methods. The "encoder" subsumes all layers that can be shared between both branches (*e.g.*, backbone, projection MLP [8], prototypes [7]). The components in red are those missing in SimSiam. We discuss the relations next.

Relation to SimCLR [8]. SimCLR relies on negative samples ("dissimilarity") to prevent collapsing. SimSiam can be thought of as "SimCLR without negatives".

To have a more thorough comparison, we append the prediction MLP h and stop-gradient to SimCLR.[7] Here is the ablation on our SimCLR reproduction:

SimCLR	w/ predictor	w/ pred. & stop-grad
66.5	66.4	66.0

Neither the stop-gradient nor the extra predictor is necessary or helpful for SimCLR. As we have analyzed in Sec. 5, the introduction of the stop-gradient and extra predictor is presumably a consequence of another underlying optimization problem. It is different from the contrastive learning problem, so these extra components may not be helpful.

Relation to SwAV [7]. SimSiam is conceptually analogous to "SwAV without online clustering". We build up this connection by recasting a few components in SwAV. (i) The shared prototype layer in SwAV can be absorbed into the Siamese encoder. (ii) The prototypes were weight-normalized outside of gradient propagation in [7]; we instead implement by full gradient computation [33].[8] (iii) The similarity function in SwAV is cross-entropy. With these abstractions, a highly simplified SwAV illustration is shown in Figure 3.

SwAV applies the Sinkhorn-Knopp (SK) transform [10] on the target branch (which is also symmetrized [7]). The SK transform is derived from online clustering [7]: it is the outcome of clustering the current batch subject to a balanced partition constraint. The balanced partition can avoid collapsing. Our method does not involve this transform.

We study the effect of the prediction MLP h and stop-gradient on SwAV. Note that SwAV applies stop-gradient on the SK transform, so we ablate by removing it. Here is the comparison on our SwAV reproduction:

SwAV	w/ predictor	remove stop-grad
66.5	65.2	NaN

Adding the predictor does not help either. Removing stop-gradient (so the model is trained end-to-end) leads to divergence. As a clustering-based method, SwAV is inherently an alternating formulation [7]. This may explain why stop-gradient should not be removed from SwAV.

[7] We append the extra predictor to one branch and stop-gradient to the other branch, and symmetrize this by swapping.

[8] This modification produces similar results as original SwAV, but it can enable end-to-end propagation in our ablation.

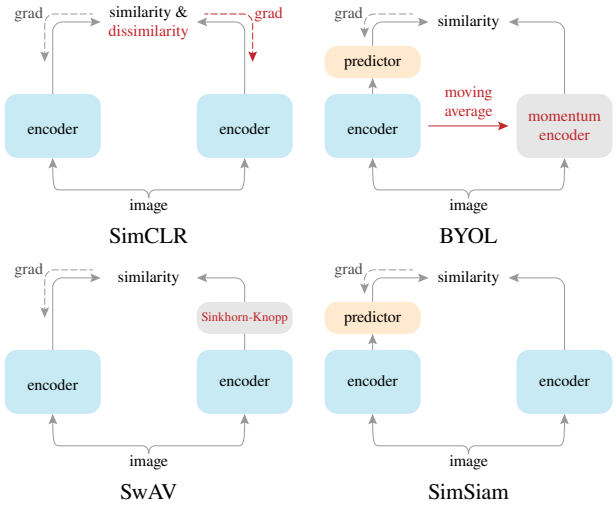


Figure 3. **Comparison on Siamese architectures**. The encoder includes all layers that can be shared between both branches. The dash lines indicate the gradient propagation flow. In BYOL, SwAV, and SimSiam, the lack of a dash line implies stop-gradient, and their symmetrization is not illustrated for simplicity. The components in red are those missing in SimSiam.

Relation to BYOL [15]. Our method can be thought of as "BYOL without the momentum encoder", subject to many implementation differences. The momentum encoder may be beneficial for accuracy (Table 4), but it is not necessary for preventing collapsing. Given our hypothesis in Sec. 5, the η sub-problem (8) can be solved by other optimizers, *e.g.*, a gradient-based one. This may lead to a temporally smoother update on η. Although not directly related, the momentum encoder also produces a smoother version of η. We believe that other optimizers for solving (8) are also plausible, which can be a future research problem.

7. Conclusion

We have explored Siamese networks with simple designs. The competitiveness of our minimalist method suggests that the Siamese shape of the recent methods can be a core reason for their effectiveness. Siamese networks are natural and effective tools for modeling invariance, which is a focus of representation learning. We hope our study will attract the community's attention to the fundamental role of Siamese networks in representation learning.

References

[1] Yuki Markus Asano, Christian Rupprecht, and Andrea Vedaldi. Self-labelling via simultaneous clustering and representation learning. *arXiv:1911.05371*, 2019.

[2] Philip Bachman, R Devon Hjelm, and William Buchwalter. Learning representations by maximizing mutual information across views. *arXiv:1906.00910*, 2019.

[3] Luca Bertinetto, Jack Valmadre, Joao F Henriques, Andrea Vedaldi, and Philip HS Torr. Fully-convolutional Siamese networks for object tracking. In *ECCV*, 2016.

[4] Jane Bromley, Isabelle Guyon, Yann LeCun, Eduard Säckinger, and Roopak Shah. Signature verification using a "Siamese" time delay neural network. In *NeurIPS*, 1994.

[5] Mathilde Caron, Piotr Bojanowski, Armand Joulin, and Matthijs Douze. Deep clustering for unsupervised learning of visual features. In *ECCV*, 2018.

[6] Mathilde Caron, Piotr Bojanowski, Julien Mairal, and Armand Joulin. Unsupervised pre-training of image features on non-curated data. In *ICCV*, 2019.

[7] Mathilde Caron, Ishan Misra, Julien Mairal, Priya Goyal, Piotr Bojanowski, and Armand Joulin. Unsupervised learning of visual features by contrasting cluster assignments. *arXiv:2006.09882*, 2020.

[8] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. *arXiv:2002.05709*, 2020.

[9] Xinlei Chen, Haoqi Fan, Ross Girshick, and Kaiming He. Improved baselines with momentum contrastive learning. *arXiv:2003.04297*, 2020.

[10] Marco Cuturi. Sinkhorn distances: Lightspeed computation of optimal transport. In *NeurIPS*, 2013.

[11] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. ImageNet: A large-scale hierarchical image database. In *CVPR*, 2009.

[12] Mark Everingham, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The PASCAL Visual Object Classes (VOC) Challenge. *IJCV*, 2010.

[13] Ross Girshick, Ilija Radosavovic, Georgia Gkioxari, Piotr Dollár, and Kaiming He. Detectron, 2018.

[14] Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch SGD: Training ImageNet in 1 hour. *arXiv:1706.02677*, 2017.

[15] Jean-Bastien Grill, Florian Strub, Florent Altché, Corentin Tallec, Pierre H. Richemond, Elena Buchatskaya, Carl Doersch, Bernardo Avila Pires, Zhaohan Daniel Guo, Mohammad Gheshlaghi Azar, Bilal Piot, Koray Kavukcuoglu, Rémi Munos, and Michal Valko. Bootstrap your own latent: A new approach to self-supervised learning. *arXiv:2006.07733v1*, 2020.

[16] Raia Hadsell, Sumit Chopra, and Yann LeCun. Dimensionality reduction by learning an invariant mapping. In *CVPR*, 2006.

[17] Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross Girshick. Momentum contrast for unsupervised visual representation learning. *arXiv:1911.05722*, 2019.

[18] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask R-CNN. In *ICCV*, 2017.

[19] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016.

[20] R Devon Hjelm, Alex Fedorov, Samuel Lavoie-Marchildon, Karan Grewal, Adam Trischler, and Yoshua Bengio. Learning deep representations by mutual information estimation and maximization. In *ICLR*, 2019.

[21] Olivier J. Hénaff, Aravind Srinivas, Jeffrey De Fauw, Ali Razavi, Carl Doersch, S. M. Ali Eslami, and Aaron van den Oord. Data-efficient image recognition with contrastive predictive coding. *arXiv:1905.09272v2*, 2019.

[22] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, 2015.

[23] Gregory Koch, Richard Zemel, and Ruslan Salakhutdinov. Siamese neural networks for one-shot image recognition. In *ICML deep learning workshop*, 2015.

[24] Alex Krizhevsky. Learning multiple layers of features from tiny images. *Tech Report*, 2009.

[25] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1989.

[26] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft COCO: Common objects in context. In *ECCV*. 2014.

[27] Ilya Loshchilov and Frank Hutter. SGDR: Stochastic gradient descent with warm restarts. In *ICLR*, 2017.

[28] James MacQueen et al. Some methods for classification and analysis of multivariate observations. 1967.

[29] Ishan Misra and Laurens van der Maaten. Self-supervised learning of pretext-invariant representations. *arXiv:1912.01991*, 2019.

[30] Aaron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding. *arXiv:1807.03748*, 2018.

[31] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. PyTorch: An imperative style, high-performance deep learning library. In *NeurIPS*, 2019.

[32] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN: Towards real-time object detection with region proposal networks. In *NeurIPS*, 2015.

[33] Tim Salimans and Diederik P Kingma. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. In *NeurIPS*, 2016.

[34] Yaniv Taigman, Ming Yang, MarcAurelio Ranzato, and Lior Wolf. DeepFace: Closing the gap to human-level performance in face verification. In *CVPR*, 2014.

[35] Yonglong Tian, Dilip Krishnan, and Phillip Isola. Contrastive multiview coding. *arXiv:1906.05849*, 2019.

[36] Zhirong Wu, Yuanjun Xiong, Stella Yu, and Dahua Lin. Unsupervised feature learning via non-parametric instance discrimination. In *CVPR*, 2018.

[37] Mang Ye, Xu Zhang, Pong C Yuen, and Shih-Fu Chang. Unsupervised embedding learning via invariant and spreading instance feature. In *CVPR*, 2019.

[38] Yang You, Igor Gitman, and Boris Ginsburg. Large batch training of convolutional networks. *arXiv:1708.03888*, 2017.

A. Implementation Details

Unsupervised pre-training. Our implementation follows the practice of existing works [36, 17, 8, 9, 15].

Data augmentation. We describe data augmentation using the PyTorch [31] notations. Geometric augmentation is `RandomResizedCrop` with scale in $[0.2, 1.0]$ [36] and `RandomHorizontalFlip`. Color augmentation is `ColorJitter` with {brightness, contrast, saturation, hue} strength of {0.4, 0.4, 0.4, 0.1} with an applying probability of 0.8, and `RandomGrayscale` with an applying probability of 0.2. Blurring augmentation [8] has a Gaussian kernel with std in $[0.1, 2.0]$.

Initialization. The convolution and fc layers follow the default PyTorch initializers. Note that by default PyTorch initializes fc layers' weight and bias by a uniform distribution $\mathcal{U}(-\sqrt{k}, \sqrt{k})$ where $k=\frac{1}{\text{in\_channels}}$. Models with substantially different fc initializers (*e.g.*, a fixed std of 0.01) may not converge. Moreover, similar to the implementation of [8], we initialize the scale parameters as 0 [14] in the last BN layer for every residual block.

Weight decay. We use a weight decay of 0.0001 for all parameter layers, including the BN scales and biases, in the SGD optimizer. This is in contrast to the implementation of [8, 15] that excludes BN scales and biases from weight decay in their LARS optimizer.

Linear evaluation. Given the pre-trained network, we train a supervised linear classifier on frozen features, which are from ResNet's global average pooling layer (pool_5). The linear classifier training uses base $lr = 0.02$ with a cosine decay schedule for 90 epochs, weight decay $= 0$, momentum $= 0.9$, batch size $= 4096$ with a LARS optimizer [38]. We have also tried the SGD optimizer following [17] with base $lr = 30.0$, weight decay $= 0$, momentum $= 0.9$, and batch size $= 256$, which gives $\sim$1% lower accuracy. After training the linear classifier, we evaluate it on the center 224×224 crop in the validation set.

B. Additional Ablations on ImageNet

The following table reports the SimSiam results *vs*. the output dimension d:

output d	256	512	1024	2048
acc. (%)	65.3	67.2	67.5	68.1

It benefits from a larger d and gets saturated at $d = 2048$. This is unlike existing methods [36, 17, 8, 15] whose accuracy is saturated when d is 256 or 512.

In this table, the prediction MLP's hidden layer dimension is always 1/4 of the output dimension. We find that this bottleneck structure is more robust. If we set the hidden dimension to be equal to the output dimension, the training can be less stable or fail in some variants of our exploration. We hypothesize that this bottleneck structure, which behaves like an auto-encoder, can force the predictor to digest the information. We recommend to use this bottleneck structure for our method.

	SimCLR			MoCo v2		BYOL			SwAV
epoch	200	800	1000	200	800	300	800	1000	400
origin	66.6	68.3	69.3	67.5	71.1	72.5	-	74.3	70.1
repro.	68.3	70.4	-	69.9	72.2	72.4	74.3	-	70.7

Table C.1. **Our reproduction *vs*. original papers' results**. All are based on ResNet-50 pre-trained with two 224×224 crops.

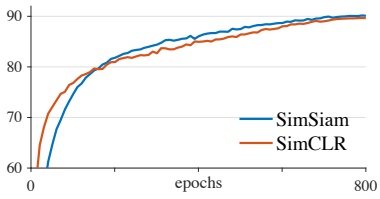
C. Reproducing Related Methods

Our comparison in Table 4 is based on our reproduction of the related methods. We re-implement the related methods as faithfully as possible following each individual paper. In addition, we are able to improve SimCLR, MoCo v2, and SwAV by small and straightforward modifications: specifically, we use 3 layers in the projection MLP in SimCLR and SwAV (*vs*. originally 2), and use symmetrized loss for MoCo v2 (*vs*. originally asymmetric). Table C.1 compares our reproduction of these methods with the original papers' results (if available). Our reproduction has *better* results for SimCLR, MoCo v2, and SwAV (denoted as "+" in Table 4), and has at least comparable results for BYOL.

D. CIFAR Experiments

We have observed similar behaviors of SimSiam in the CIFAR-10 dataset [24]. The implementation is similar to that in ImageNet. We use SGD with base $lr = 0.03$ and a cosine decay schedule for 800 epochs, weight decay $= 0.0005$, momentum $= 0.9$, and batch size $= 512$. The input image size is 32×32. We do not use blur augmentation. The backbone is the CIFAR variant of ResNet-18 [19], followed by a 2-layer projection MLP. The outputs are 2048-d.

Figure D.1 shows the kNN classification accuracy (left) and the linear evaluation (right). Similar to the ImageNet observations, SimSiam achieves a reasonable result and does not collapse. We compare with SimCLR [8] trained with the same setting. Interestingly, the training curves are similar between SimSiam and SimCLR. SimSiam is slightly better by 0.7% under this setting.



	acc. (%)
SimCLR	91.1
SimSiam	91.8

Figure D.1. **CIFAR-10 experiments**. Left: validation accuracy of kNN classification as a monitor during pre-training. Right: linear evaluation accuracy. The backbone is ResNet-18.