# Exploring Simple Siamese Representation Learning

Seri Lee
*Computer Science and Engineering*
*Seoul National University*
Seoul, Republic of Korea
sally20921@snu.ac.kr

***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 cerrtain 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: (1) negative sample pairs, (2) large batches, (3) 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.**

## I. Introduction

Recently there has been steady progress in self-supervised representation learning, with encouraging results on multiple visual tasks [**?**]. Despite various original motivations, these methods generally involve certain forms of Siamese networks. Siamese networks are weight-sharing neural networks applied on two or more inputs. They are natural tool s 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 network is all outputs "collapsing" to a constant. There have been several general strategies for preventing Siamese networks from collapsing. Contrastive learning 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 is another way of avoiding constant output, and SwAV incorporates online clustering into Siamese networks. Beyond contrastive learning and clustering, BYOL relies only on positive pairs but it does not collapse in case a momentum encoder is used.

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.

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 be thought of as "SimCLR without negative pairs" and "SwAV without online clustering". Interestingly, SimSiam is related to each nethod 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 is critical to prevent such solutions. The importance of stop-gradient suggests that thereshould be a different underlying optimization problem that is being solved. We hypothesize taht there are implicitly two sets of variables, and SimSiam behaves like alternating between optimzing each set. We provide proof-of -concept experiments to verify this hypothesis.

Our simple baseline suggests that the Siamese architectures can be essential reason for the common success of the related methods. Siamese network 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, which is a successful inductive bias via weight-sharing for modeling tanslation-invariance, the weight-sharing Siamese networks can model invariance w.r.t more complicated transformations (e.g., augmentations). We hope that our exploration will motivate people to rethink the fundamental roles of Siamese architectures for unsupervised representation learning.

## II. Related Works

**Siamese networks** Siamese networks are general models for comparing entities. Their applications include signature and face verification, tracking, one-shot learning, 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 is to attract the positive sample pairs and repulse the negative sample pairs. This methodology has been recently popularized for unsupervised representation learning. Simple and effective instantiations of contrastive learning have been developed using Siamese networks.

In practice, contrastive learning methods benefit from a large number of negative samples. These samples can be amintained in a memory bank. In a Siamese network, MoCo maintains a queue of negative samples and turns one branch

into a momentum encoder to improve consistency of the queue. SimCLR directly uses negative samples coexisting in the current batch, and it requires a large batch to work well.

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

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

**BYOL** BYOL directly predicts the output of one view from another view. It is a Siamese network in which one branch is a momentum encoder. It is hypothesized in that the momentum encoder is important for BYOL to avoid collapsing, and it reports failure results if removing the momentum encoder. 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. 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.

## III. Method

Our architecture 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) and a projection MLP head. The encoder $f$ shares weights between the two views. A prediction MLP head, 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:

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

where $\|\cdot\|_2$ is $l_2$-norm. This is equivalent to the mean squared error of $l_2$-normalized vectors, up to a scale of 2. We define a symmetrized loss as:

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

. 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. We implement it by modifying as:

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

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

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

Here the encode 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. We use a learning rate of $lr \times BatchSize/256$ (linear scaling), with a base $lr = 0.05$. The learning rate has a cosine decay schedule. 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. We use batch normalization (BN) synchronized across devices.
- *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 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 diemnsion is 512, making $h$ a bottleneck structure.

We use ResNet-50 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 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.

## IV. 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.

### A. Stop-gradient

Without stop-gradient, the optimizer quickly finds a degnerated 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.

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

With stop-gradient, the kNN monitor shows a steadily improving accuracy.

**Discussion** Our experiments show that *there exist collapsing solutions*. The collapse can be observed by the minimum possible loss and the constant outputs. 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.

### B. Predictor

We study the predictor MLP's effect. The model does not work if removing $h$, *i.e.*, $h$ is the identity mapping. Actually, this observation can be expected if the symmetric loss is used. Now the loss is $\frac{1}{2}D(z_1, stopgrad(z_2)) + D(z_2, stopgrad(z_1))$. Its gradient has the same direction as the gradient of $D(z_1, z_2)$, 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.

We note that this derivation on the gradient direction is valid only for the symmetrized loss. These experiments suggest that $h$ is helpful for our model.

If $h$ is fixed as random initialization, our model does not work either. 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. 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.

### C. 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 ($lr \times BatchSize/256$) with the base $lr = 0.05$. We use 10 epochs of warm-up for batch sizes
$geq1024$. Note that we keep using the same SGD optimizer (rather than LARS) 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 bath 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 and SwAV. 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), and our result is lower with a 4096 batch. We expect a specialized optimizer (*e.g.* LARS) will help in this case. However, our results show that a specialized optimizer is not necessary for preventing collapsing.

### D. Batch Normalization

Table 3 comares the configurations of BN on the MLP heads. The low accuracy is likely because of optimization difficulty. Adding BN to the hidden layers increases the accuracy to 67.4%.

Further adding BN to the output of the projection MLP boosts accuracy to 68.1%, which is our default configuration. In this entry, we also find that the learnable affine transformation (scale and offset) 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. 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.

### E. Similarity Function

Besides the cosine similarity function, our method also works well with cross-entropy similarity. We modify $D$ as: $D(p_1, z_2) = -softmax(z_2) \cdot \log softmax(p_1)$. 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. All hyper-parameters and architectures are unchanged, though they may be suboptimal for this variant.

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.

### F. Symmetrization