D. Representation Learning

D.1 Representation Learning: Learning useful coordinates for data

D.2 Representation Learning: Evaluating and probing Representations

D.3 Representation Learning: How to learn a good Representation

In this lecture we will learn about:

(i) The history of learning by shaping distances

(ii) Metric learning

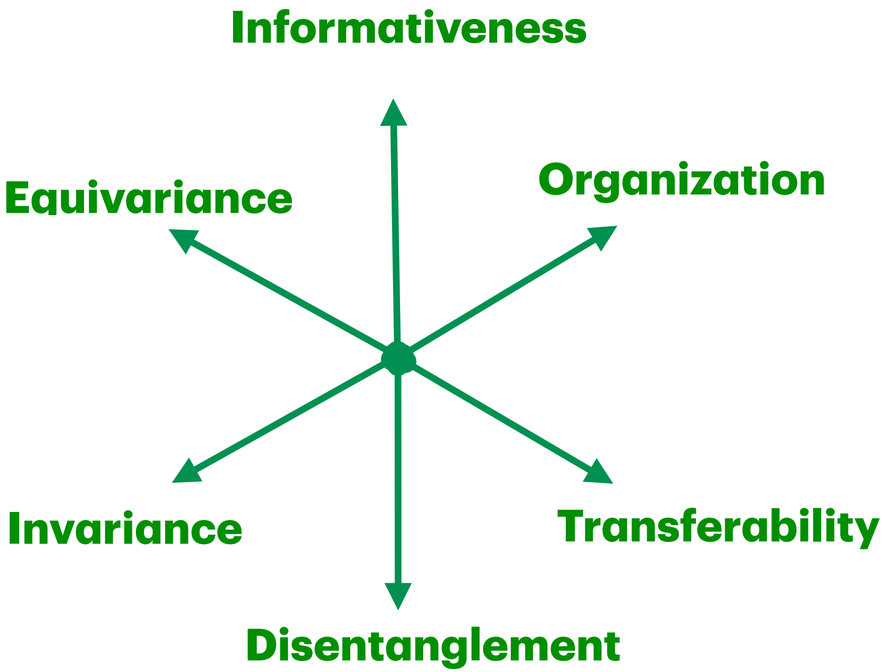
(iii) Contrastive Learning

we learned about some attributes and measures of a good* representation:

Attributes

1. **Sufficient**: keeps task-relevant information
2. **Compact**: removes irrelevant variation
3. **Structured**: similar examples are nearby
4. **Separated**: task-relevant groups are easier to distinguish
5. **Robust**: ignores irrelevant perturbations, but preserves meaningful differences

measures



In this lecture we will learn about some creative ways to learn a good representation

*NeurIPS 2020 Competition: Predicting Generalization in Deep Learning (Version 1.1)

(i) Introduction: Why do we want to learn a good representation?

Raw data x is often high-dimensional, noisy, and hard to compare directly. We learn an encoder so that useful structure becomes easier to access in representation space:

$$z = f_\theta(x)$$

A good representation helps us to:

1. Generalize better: A simpler downstream model can solve the task if z already organizes the relevant information:

$$\hat{y} = g_\phi(z)$$

2. Transfer to new tasks: A representation learned once can be reused for new datasets, labels, or objectives.

3. Compare new examples semantically:

$$d(z_i, z_j) \approx \text{semantic difference between } x_i \text{ and } x_j$$

This enables face verification, retrieval, nearest-neighbor search, and query-based matching.

4. Cluster and organize data: Representations can group similar examples (ideally linearly separable), especially when we have side information such as:

$$(x_i, x_j) \text{ similar} \qquad (x_i, x_k) \text{ dissimilar}$$

5. Reduce dimensionality: We want a compact space that preserves the important structure of the data.

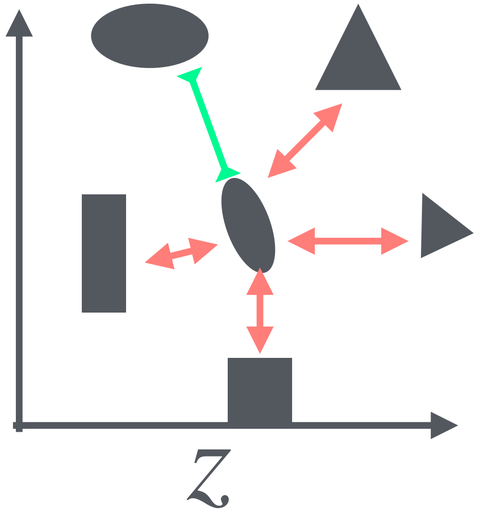
(i) Introduction: Learning by shaping distances

Earlier, we asked what makes a representation ($z = f_\theta(x)$) good. A simple idea is:

If two inputs are similar : $x_i \sim x_j$ $\Rightarrow$ $d(z_i, z_j)$ should be small

If two inputs are different: $x_i \nsim x_k$ $\Rightarrow$ $d(z_i, z_k)$ should be large

In a way this idea works as set of **attractive** (between similar inputs)and **repulsive** (between dissimilar inputs) forces in the latent space Z that leads to re-organizations of Z:



Suggestion: Instead of only supervising the final prediction, give feedback directly on the representation space z.

This is the core idea behind **metric learning**.

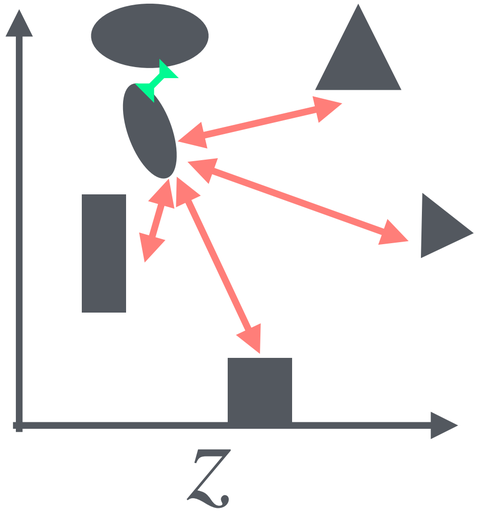
(i) Introduction: Learning by shaping distances

Earlier, we asked what makes a representation ($z = f_\theta(x)$) good. A simple idea is:

If two inputs are similar : $x_i \sim x_j$ $\Rightarrow$ $d(z_i, z_j)$ should be small

If two inputs are different: $x_i \nsim x_k$ $\Rightarrow$ $d(z_i, z_k)$ should be large

In a way this idea works as set of **attractive** (between similar inputs)and **repulsive** (between dissimilar inputs) forces in the latent space Z that leads to re-organizations of Z:



Suggestion: Instead of only supervising the final prediction, give feedback directly on the representation space z.

This is the core idea behind **metric learning**.

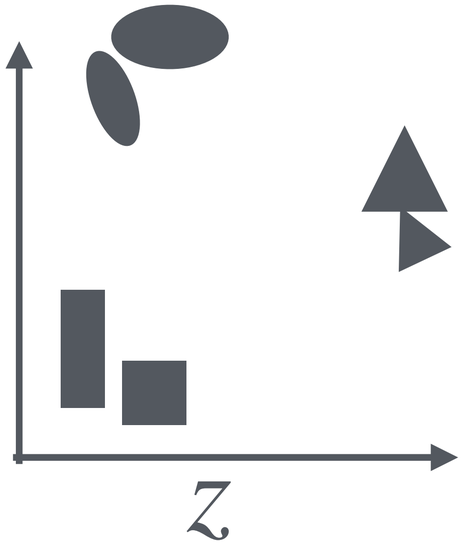
(i) Introduction: Learning by shaping distances

Earlier, we asked what makes a representation ($z = f_\theta(x)$) good. A simple idea is:

If two inputs are similar : $x_i \sim x_j$ $\Rightarrow$ $d(z_i, z_j)$ should be small

If two inputs are different: $x_i \nsim x_k$ $\Rightarrow$ $d(z_i, z_k)$ should be large

In a way this idea works as set of **attractive** (between similar inputs)and **repulsive** (between dissimilar inputs) forces in the latent space Z that leads to re-organizations of Z.
Until we reach equilibrium!



Suggestion: Instead of only supervising the final prediction, give feedback directly on the representation space z.

This is the core idea behind **metric learning**.

(i) Introduction: History of Learning by shaping distances (firs appearance)

Self-organizing neural network that discovers surfaces in random-dot stereograms

Suzanna Becker & Geoffrey E. Hinton

Department of Computer Science, University of Toronto,
10 King's College Road, Toronto M5S 1A4, Canada

THE standard form of back-propagation learning[1] is implausible as a model of perceptual learning because it requires an external teacher to specify the desired output of the network. We show how the external teacher can be replaced by internally derived teaching signals. These signals are generated by using the assumption that different parts of the perceptual input have common causes in the external world. Small modules that look at separate but related parts of the perceptual input discover these common causes by striving to produce outputs that agree with each other (Fig. 1*a*). The modules may look at different modalities (such as vision and touch), or the same modality at different times (for example, the consecutive two-dimensional views of a rotating three-dimensional object), or even spatially adjacent parts of the same image. Our simulations show that when our learning procedure is applied to adjacent patches of two-dimensional images, it allows a neural network that has no prior knowledge of the third dimension to discover depth in random dot stereograms of curved surfaces.

NATURE · VOL 355 · 9 JANUARY 1992

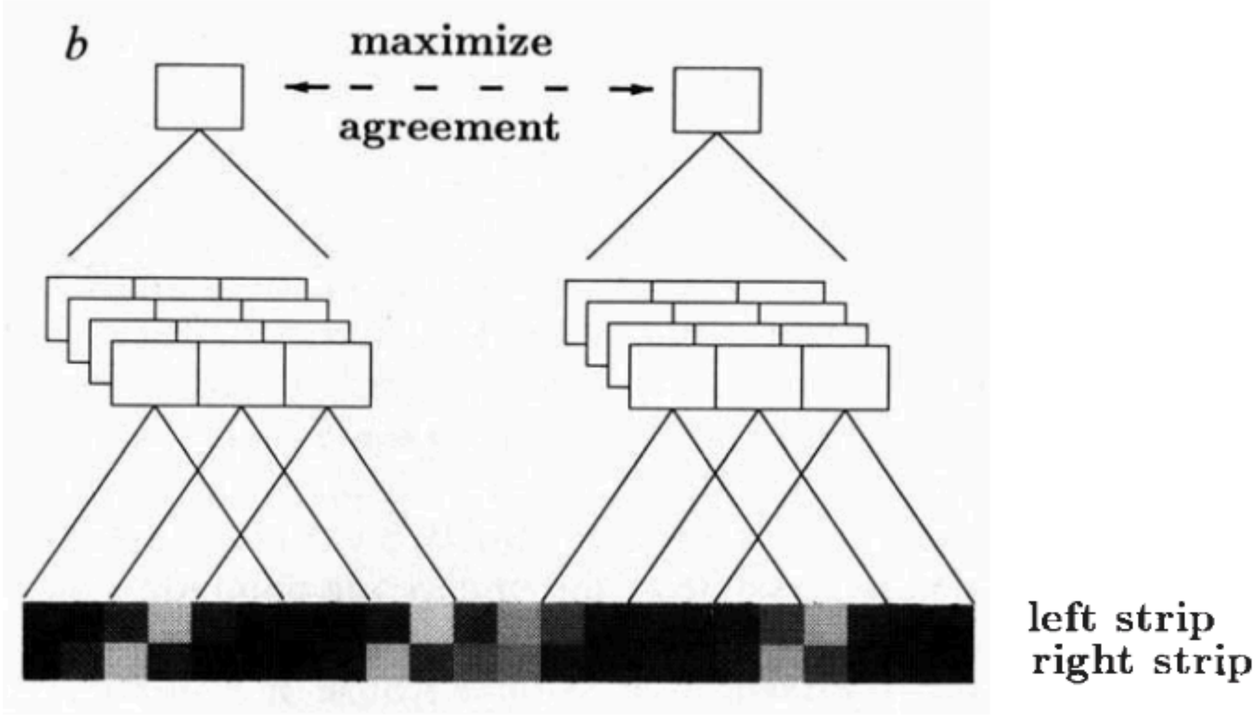
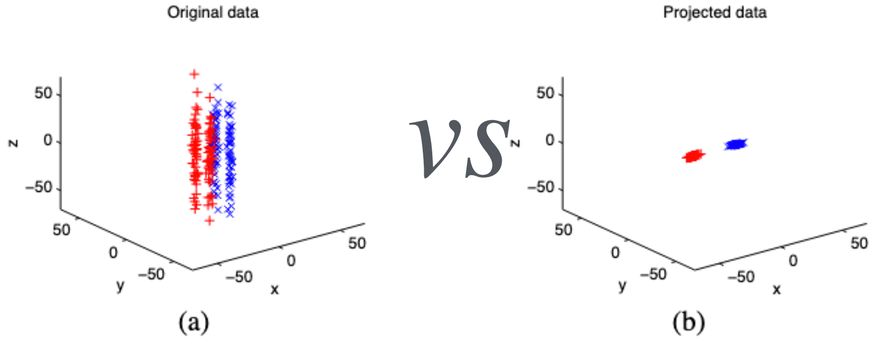


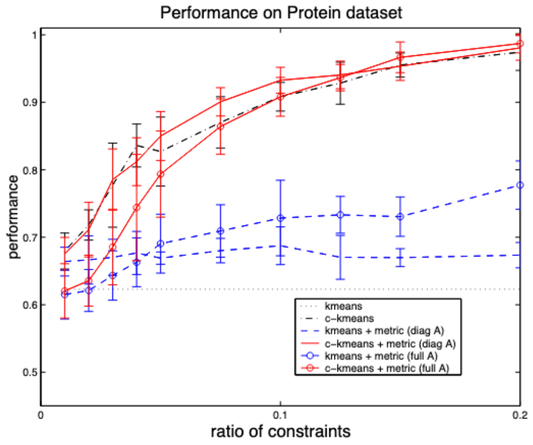
FIG. 1 *a*, Two modules that look at separate but related parts of the perceptual input and attempt to produce the same output. By differentiating

Becker, Suzanna, and Geoffrey E. Hinton. "Self-organizing neural network that discovers surfaces in random-dot stereograms." Nature 355.6356 (1992): 161-163.

(i) Introduction: History of Learning by shaping distances (the linear model)

Historically, one natural way to learn good representations was not to define goodness only through a downstream classifier, but to give feedback directly in the embedding space. If two examples are known to be similar, their representations should move closer. If they are dissimilar, their representations should move apart. This is the basic idea behind metric learning.





This approach can potentially improve these attributes:

1. **Sufficient**: keeps task-relevant information
2. **Compact**: removes irrelevant variation
3. **Structured**: similar examples are nearby
4. **Separated**: task-relevant groups are easier to distinguish
5. **Robust**: ignores irrelevant perturbations, but preserves meaningful differences

Distance Metric Learning, with Application to Clustering with Side-Information

Eric P. Xing, Andrew Y. Ng, Michael I. Jordan and Stuart Russell
University of California, Berkeley
Berkeley, CA 94720
{epxing,ang,jordan,russell}@cs.berkeley.edu

Xing, Eric, et al. "Distance metric learning with application to clustering with side-information." *Advances in neural information processing systems* 15 (2002).

(ii) Metric learning (deep metric learning, pair loss)

We define an encoder:

$$z = f_\theta(x)$$

For a pair of inputs (x_i, x_j) we define the distance between latent points:

$$d(z_i, z_j) = \|z_i - z_j\|$$

And we can define the loss function:

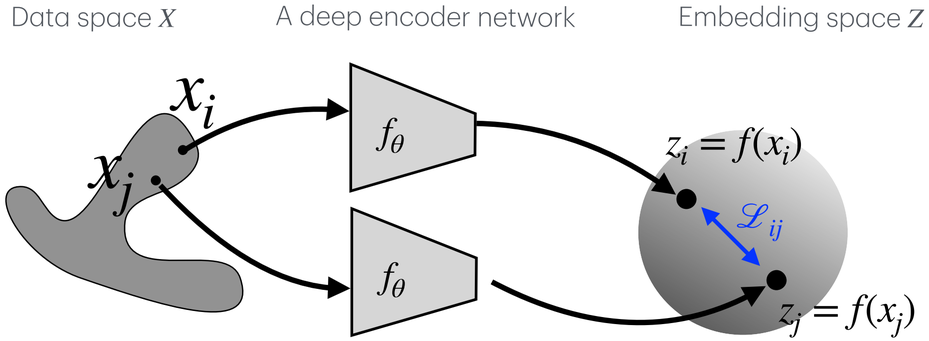
$$\mathcal{L}_{ij} = s_{ij}\, d(z_i, z_j)^2 + (1 - s_{ij})\ \max(0, m - d(z_i, z_j))^2$$

Where m is the "margin of tolerance" and:

$s_{ij} = 1$: (x_i, x_j) are in same category we say

$s_{ij} = 0$: (x_i, x_j) are not in same category

This is called " **contrastive pair loss**"



$$s_{ij} = 1: \quad \mathcal{L}_{ij} = d(z_i, z_j)^2$$

$$s_{ij} = 0: \quad \mathcal{L}_{ij} = \max(0, m - d(z_i, z_j))^2$$

The loss shapes the geometry of latent space as well, with similar pairs close, dissimilar pairs separated.

Learning a Similarity Metric Discriminatively, with Application to Face Verification

Sumit Chopra Raia Hadsell Yann LeCun

Courant Institute of Mathematical Sciences
New York University
New York, NY, USA
{sumit, raia, yann}@cs.nyu.edu

2005

(ii) Metric learning (deep metric learning, triple loss)

contrastive triple loss: Triplet loss is a later/ alternative metric-learning setup:

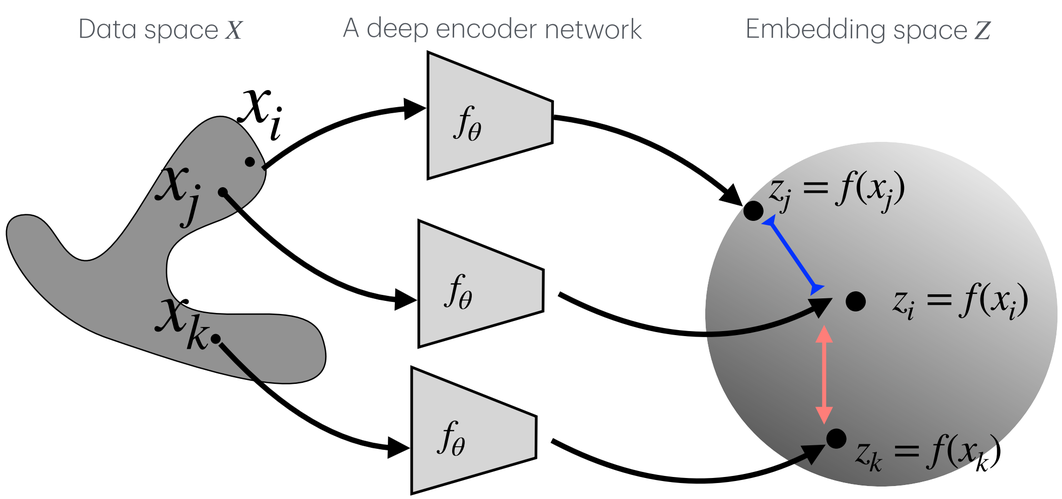
$$(x_i, x_j, x_k)$$

where :

x_i is the anchor

x_j is the positive example (similar to x_i)

x_k is the negative example (dissimilar to x_i)



The idea is to have a loss that forces:

$$d(z_i, z_j) + m < d(z_i, z_k)$$

(The distance between similars be less than dissimilar pairs)

So triplet loss compares relative distances:

$$\mathscr{L}_{triplet} = \max(0, d(z_i, z_j) - d(z_i, z_k) + m)$$

(ii) Metric learning (deep metric learning, triple loss)

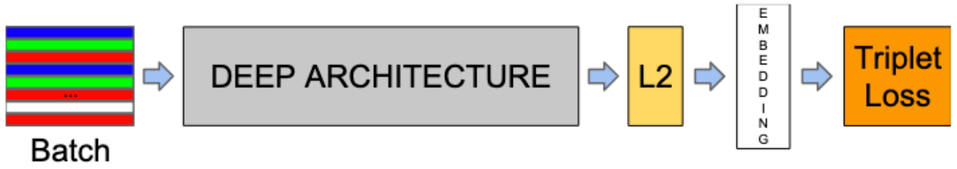


Figure 2. **Model structure.** Our network consists of a batch input layer and a deep CNN followed by L_2 normalization, which results in the face embedding. This is followed by the triplet loss during training.

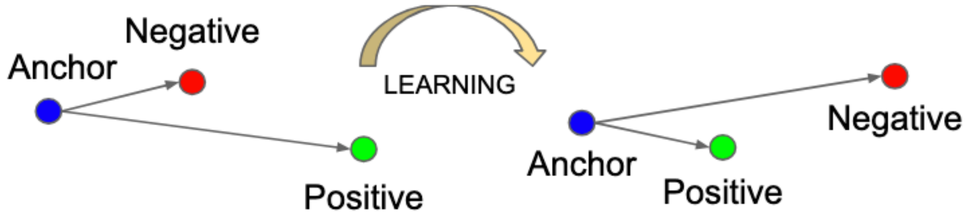


Figure 3. The **Triplet Loss** minimizes the distance between an *anchor* and a *positive*, both of which have the same identity, and maximizes the distance between the *anchor* and a *negative* of a different identity.

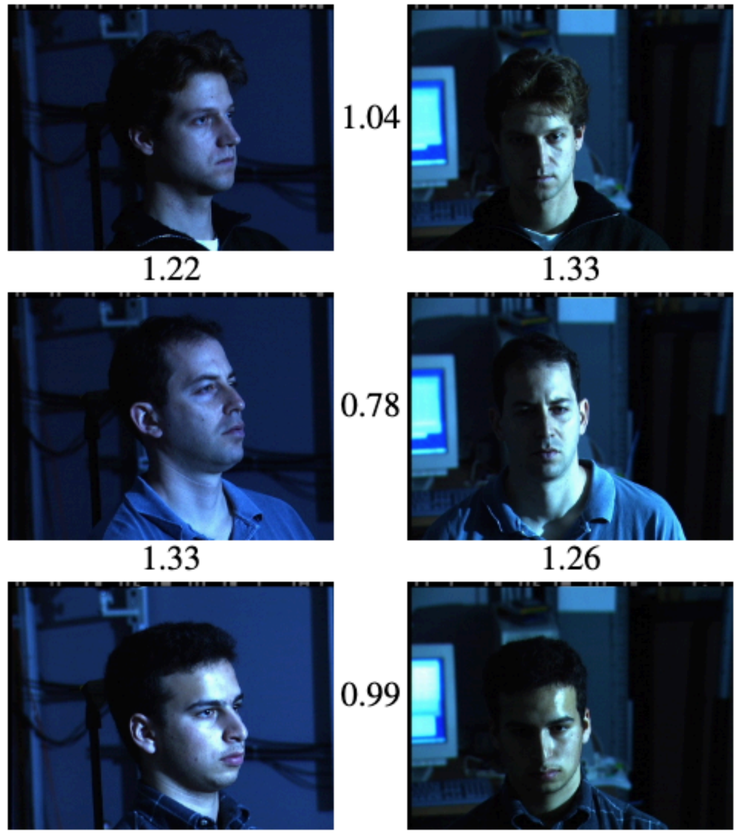


Figure 1. **Illumination and Pose invariance.** Pose and illumination have been a long standing problem in face recognition. This figure shows the output distances of FaceNet between pairs of faces of the same and a different person in different pose and illumination combinations. A distance of 0.0 means the faces are identical, 4.0 corresponds to the opposite spectrum, two different identities. You can see that a threshold of 1.1 would classify every pair correctly.

FaceNet: A Unified Embedding for Face Recognition and Clustering

Florian Schroff
fschroff@google.com
Google Inc.

Dmitry Kalenichenko
dkalenichenko@google.com
Google Inc.

James Philbin
jphilbin@google.com
Google Inc.

(ii) Metric learning: limitations and motivation for contrastive learning

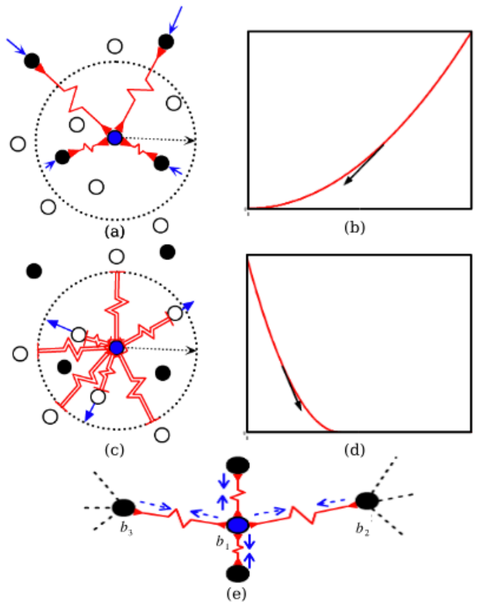
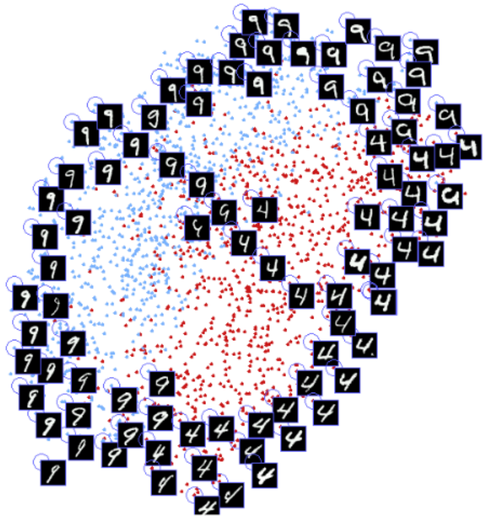


Figure 2. Figure showing the spring system. The solid circles represent points that are similar to the point in the center. The hollow circles represent dissimilar points. The springs are shown as red zigzag lines. The forces acting on the points are shown in blue arrows. The length of the arrows approximately gives the strength of the force. In the two plots on the right side, the x-axis is the distance D_W and the y-axis is the value of the loss function. (a). Shows the points connected to similar points with *attract-only* springs. (b). The loss function and its gradient associated with similar pairs. (c) The point connected only with dissimilar points inside the circle of radius m with *m-repulse-only* springs. (d) Shows the loss function and its gradient associated with dissimilar pairs. (e) Shows the situation where a point is pulled by other points in different directions, creating equilibrium.

Dimensionality Reduction by Learning an Invariant Mapping

Raia Hadsell, Sumit Chopra, Yann LeCun
The Courant Institute of Mathematical Sciences
New York University, 719 Broadway, New York, NY 1003, USA.
http://www.cs.nyu.edu/~yann
(November 2005. To appear in CVPR 2006)



However

There is a limitation:

Metric learning needs similarity information.

But in many datasets, we do not have the labels.

An alternative approach is to asks:

Can we create positive pairs automatically (self-supervised)?

And this is what contrastive learning approaches aim to do.

(iii) Contrastive Learning: motivation

Why self-supervised representation learning?

Metric learning needs pair labels, but pair labels are expensive.

Self-supervised contrastive learning asks: Can the data create its own learning signal? Instead of using human labels, we create supervision from the structure of the input itself.

- **"Pure" Reinforcement Learning** (cherry)
  - The machine predicts a scalar reward given once in a while.
  - **A few bits for some samples**
- **Supervised Learning** (icing)
  - The machine predicts a category or a few numbers for each input
  - Predicting human-supplied data
  - **10→10,000 bits per sample**
- **Unsupervised/Predictive Learning** (cake)
  - The machine predicts any part of its input for any observed part.
  - Predicts future frames in videos
  - **Millions of bits per sample**

- **(Yes, I know, this picture is slightly offensive to RL folks. But I'll make it up)**

LeCun's cake analogy ('unsupervised', 'predictive', and 'self-supervised' are used somewhat interchangeably)

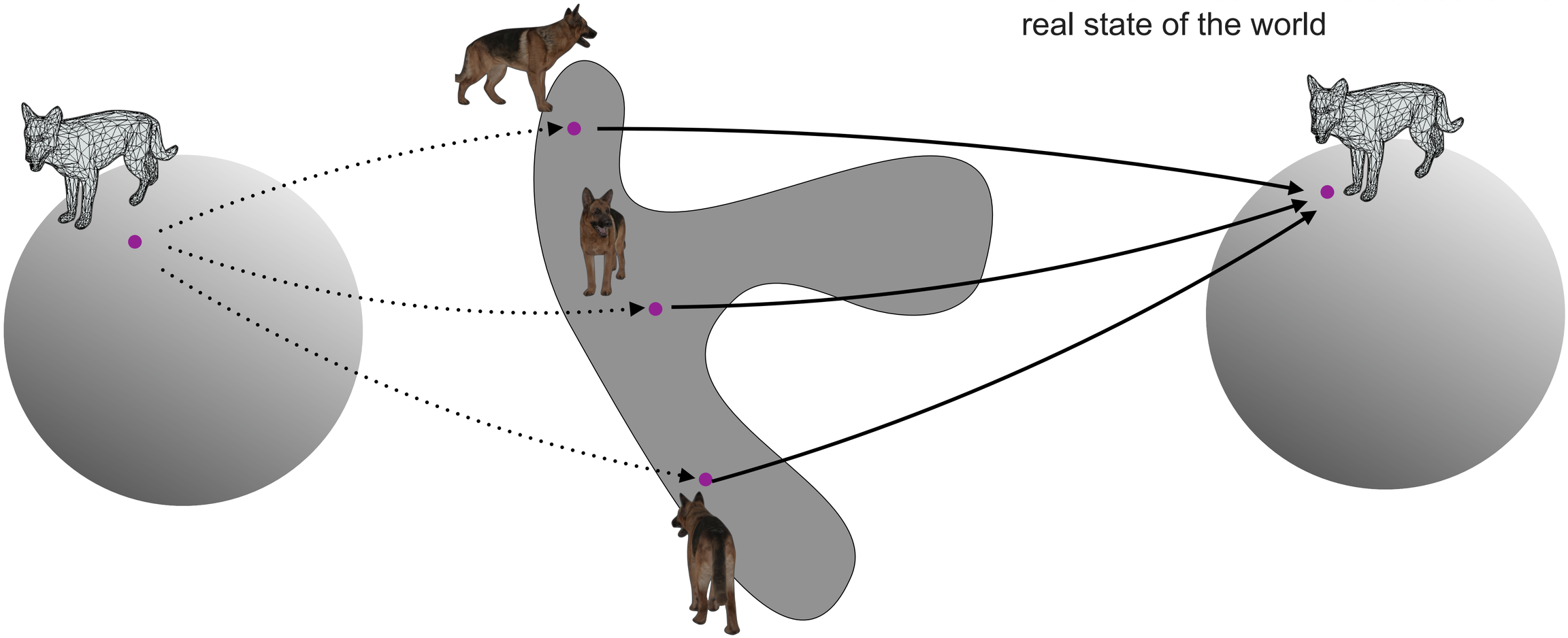
(iii) Contrastive Learning: The core idea

State: The object of this dog exists in state of the world and that animal has certain characteristics

Evidence: Our observation (data inputs) are an incomplete version of that object that exists in the state of the world

Alignment: Our goal is to align back different observations back into the state space.

Note that we don't have access to the real state of the world



https://sketchfab.com/3d-models/animated-german-sheperd-3d-animal-model-3ff58bcbea224720a54d4a2e54226859

(iii) Contrastive Learning: The core idea

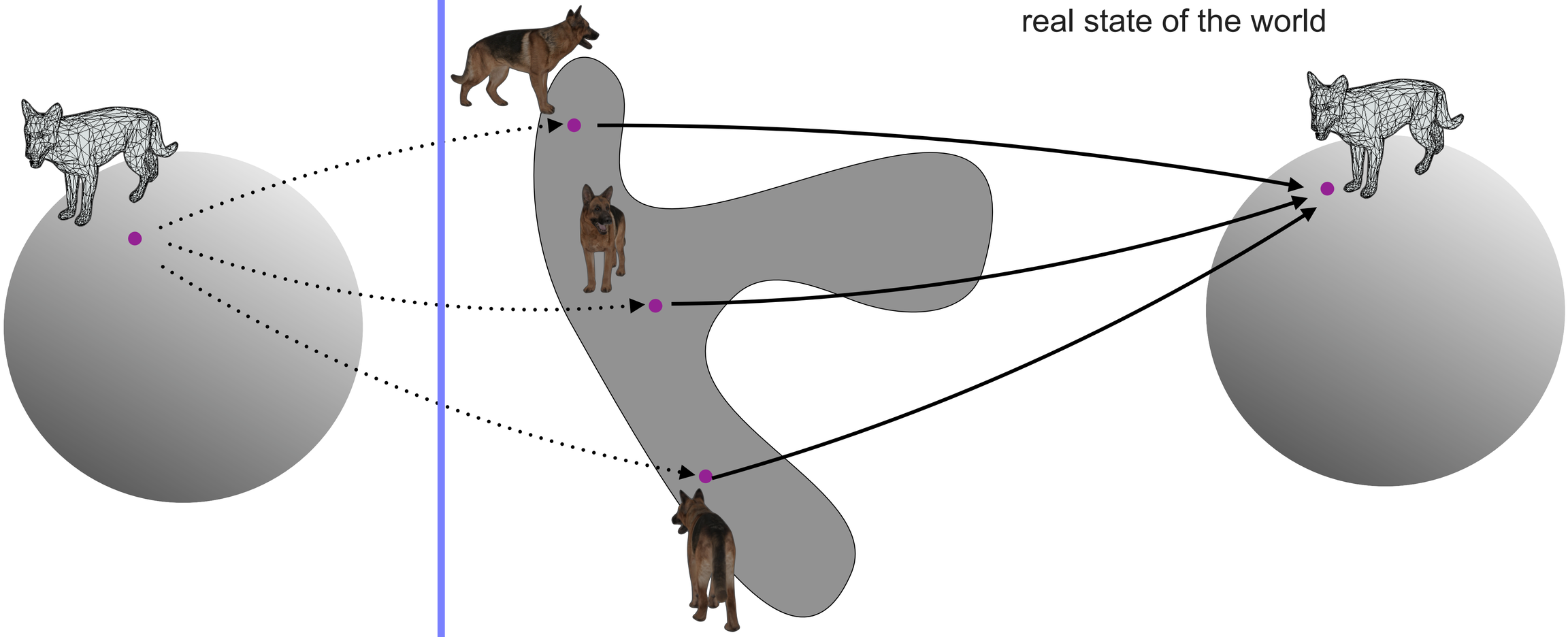
Goal of contrastive learning

State: The object of this dog exists in state of the world and that animal has certain characteristics

Evidence: Our observation (data inputs) are an incomplete version of that object that exists in the state of the world

Alignment: Our goal is to align back different observations back into the state space.

Note that we don't have access to the real state of the world



https://sketchfab.com/3d-models/animated-german-sheperd-3d-animal-model-3ff58bcbea224720a54d4a2e54226859

(iii) Contrastive Learning: Steps

We start with unlabeled input data:

$$x_1, x_2, \ldots, x_N$$

2. For each input data, we create ,multiple augmented (transformed) views

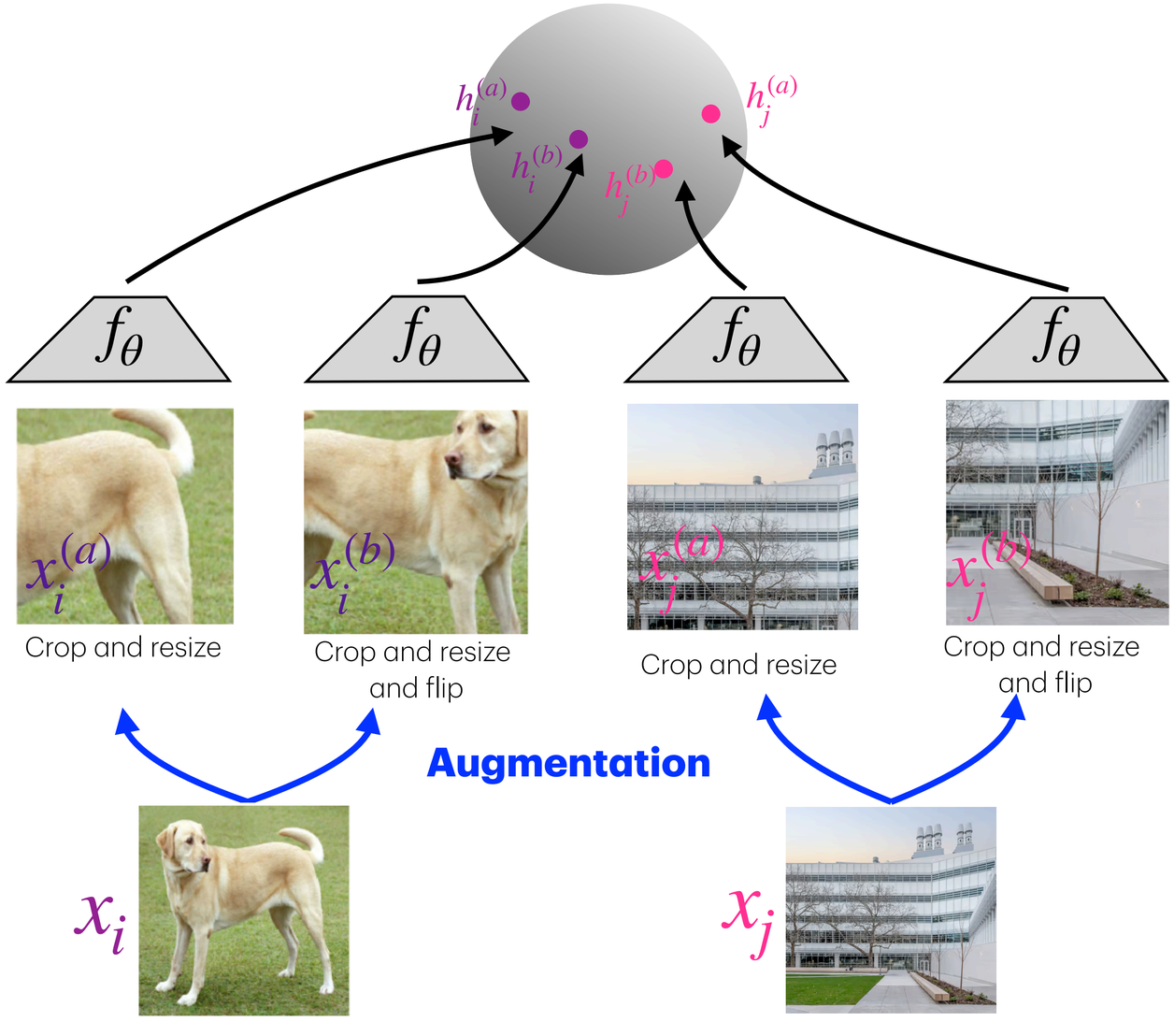
$$x_i \rightarrow x_i^{(1)}, x_i^{(2)}, \ldots, x_i^{(V)}$$

Where: $x_i^{(a)} = T_a(x_i)$, T_a is a transformation

Each view is passed through an encoder: $h_i^{(a)} = f_\theta(x_i^{(a)})$

Here h is the representation space.

Then through a MLP projection head: $z_i^{(a)} = g_\phi(h_i^{(a)})$



(iii) Contrastive Learning: Steps

We start with unlabeled input data:

$$x_1, x_2, \ldots, x_N$$

For each input data, we create ,multiple augmented (transformed) views

$$x_i \rightarrow x_i^{(1)}, x_i^{(2)}, \ldots, x_i^{(V)}$$

Where: $x_i^{(a)} = T_a(x_i)$, T_a is a transformation

Each view is passed through an encoder: $h_i^{(a)} = f_\theta(x_i^{(a)})$

Here h is the representation space.

Then through a MLP projection head: $z_i^{(a)} = g_\phi(h_i^{(a)})$

Now, similar to metric learning, for a given anchor $x_i^{(a)}$:

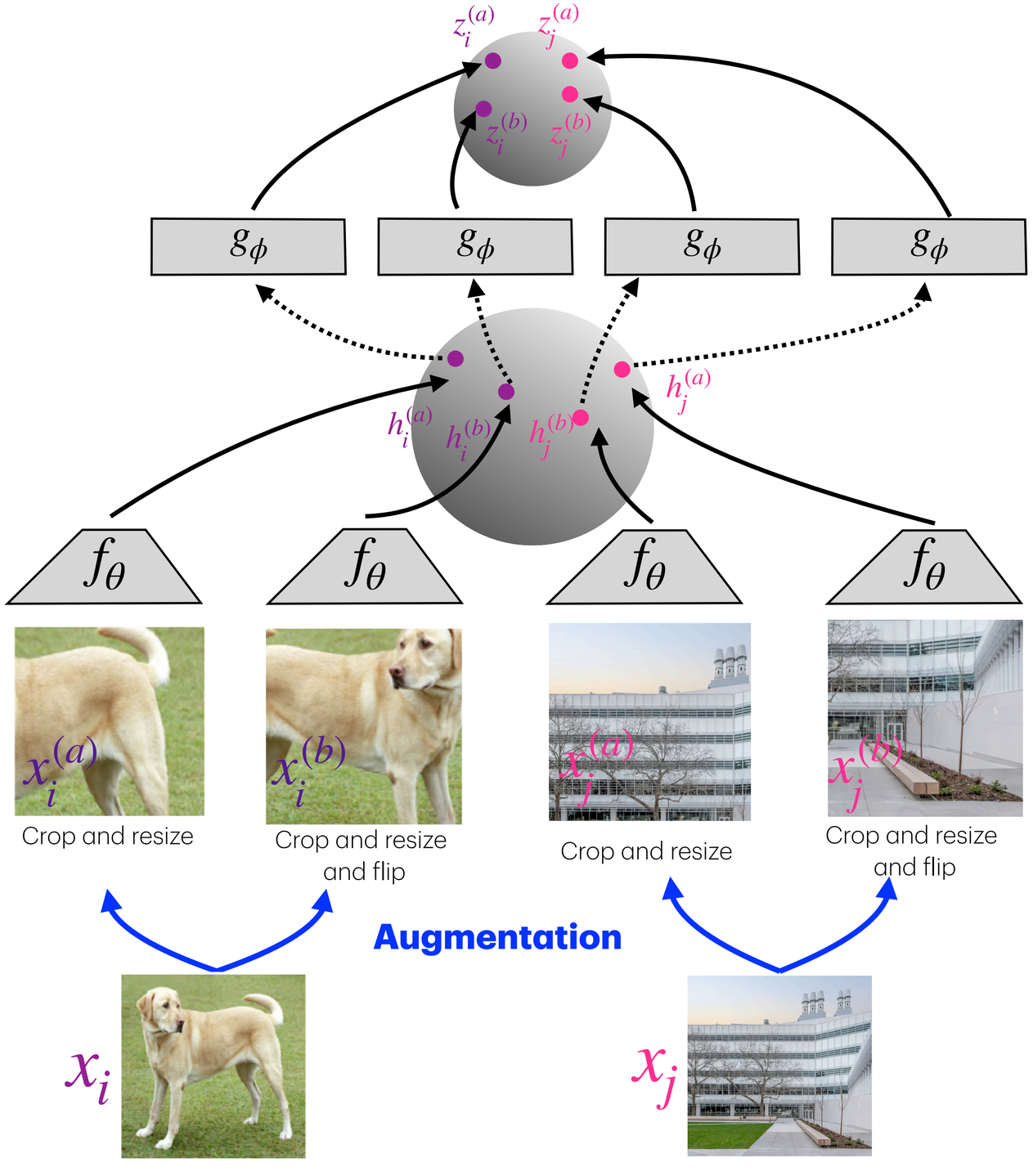
$x_i^{(b)}, b \neq a$: the positives are the other views of the same input

$x_j^{(c)}, j \neq i$: The negatives are views from different inputs

So the goal is:

$z_i^{(a)}$ close to $z_i^{(b)}$

$z_i^{(a)}$ far from $z_j^{(c)}$

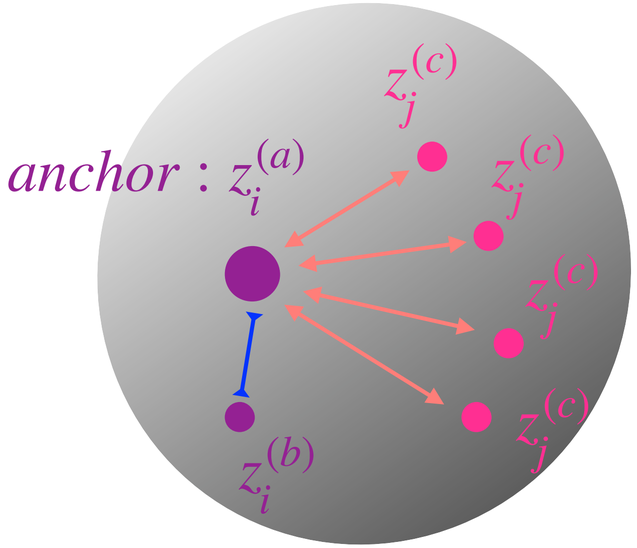


(iii) Contrastive Learning: The loss function:

First, let us set up the ideal situation:

a) For each anchor point: $(z_i^{(a)})$ we want $(z_i^{(b)}, b \neq a)$ to be close, because it is another view of the same input, positive example.

b) Also at the same time we want many negative examples $(z_j^{(c)}, j \neq i)$ to be far. (Why many negatives? Using only one negative gives a noisy learning signal)



How to achieve it:

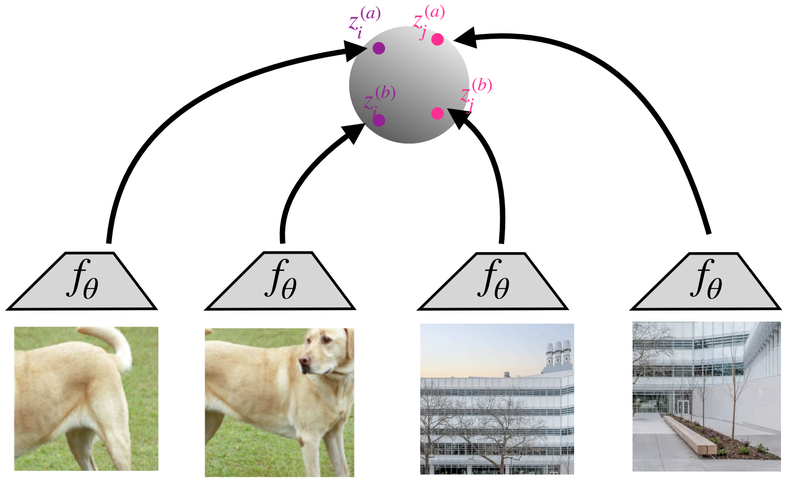
Define similarity between two points as: $sim(u, v) = \frac{u^\top v}{\|u\|\|v\|}$.

we define loss as:

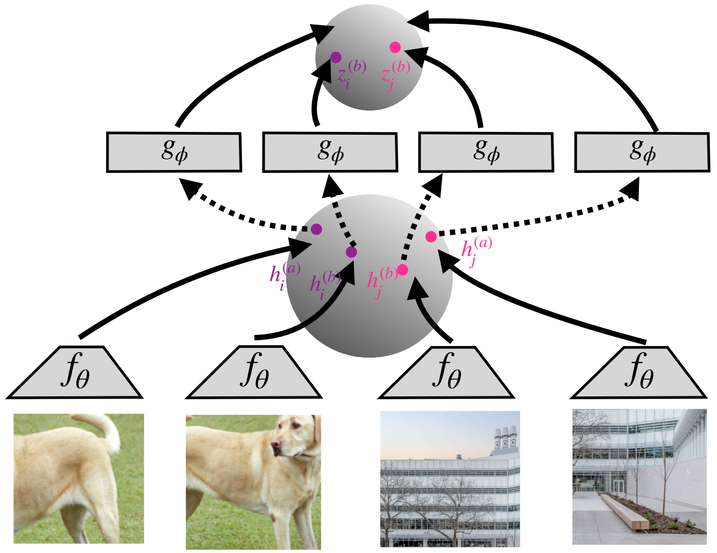
$$\mathscr{L}_{cont} = -\log \frac{e^{sim\left(z_i^{(a)}, z_i^{(b)}\right)/\tau}}{e^{sim\left(z_i^{(a)}, z_i^{(b)}\right)/\tau} + \sum_{j,c} e^{sim\left(z_i^{(a)}, z_j^{(c)}\right)/\tau}}$$

If Positive points are close to anchors (high similarity), loss decreases
If Negative points are close to anchor (high similarity), loss increases

(iii) Contrastive Learning: Why projection head?



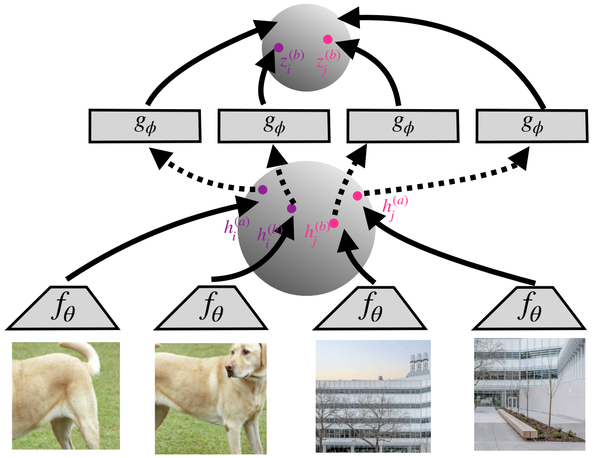
VS.



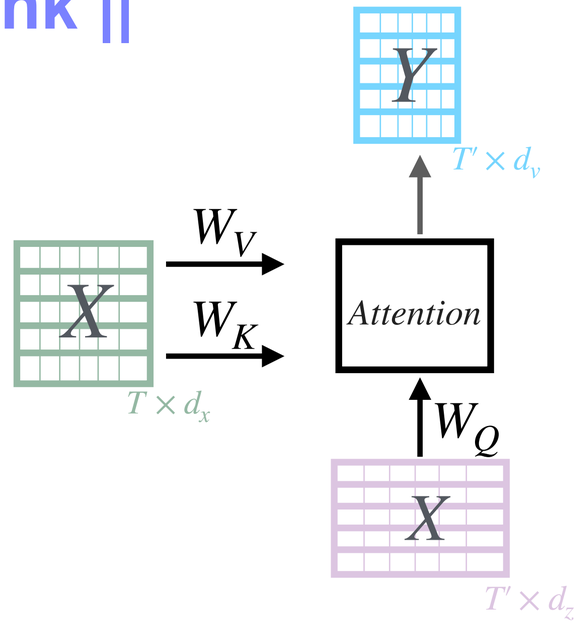
Why we don't apply contrastive loss directly to the representation layer (the output of f_θ) and add the projections head g_ϕ?

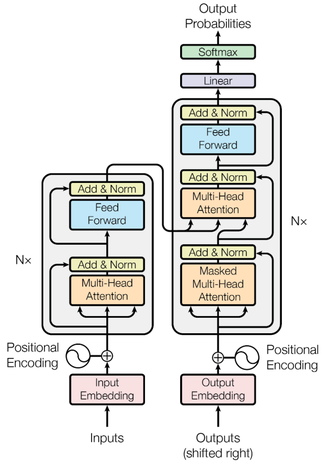
- Practically, it turned out that if we apply the contrastive loss directly on h (the encoder output), you force h to be invariant to all augmentations (color, rotation, cropping, etc.). But this destroys information that's actually useful for downstream tasks!
- By introducing a learnable nonlinear transformation g_ϕ between the representation and the contrastive loss, the model can maintain more information in h while$h = g_\phi(z)$ becomes invariant to augmentations.

A Pause point to think ||



and





Transformers use a similar concept in self-attention:In transformers, the input representations are projected into Query (**Q**), Key (**K**), and Value (**V**) spaces using learned projection matrices (Q = XW_Q, K = XW_K, V = XW_V), and the attention scores are computed in this projected space (**Q-K** space). This allows the attention mechanism to operate in a space optimized for computing similarities (**Q-K** space), while the output projection transforms results back to the residual (**V** space) stream that preserves richer information, analogous to how SimCLR's projection head creates a space optimized for contrastive loss while h preserves downstream-relevant features.

philosophy is similar: separate the space where you compute a specific operation (attention scores / contrastive loss) from the space where you preserve general-purpose representations.

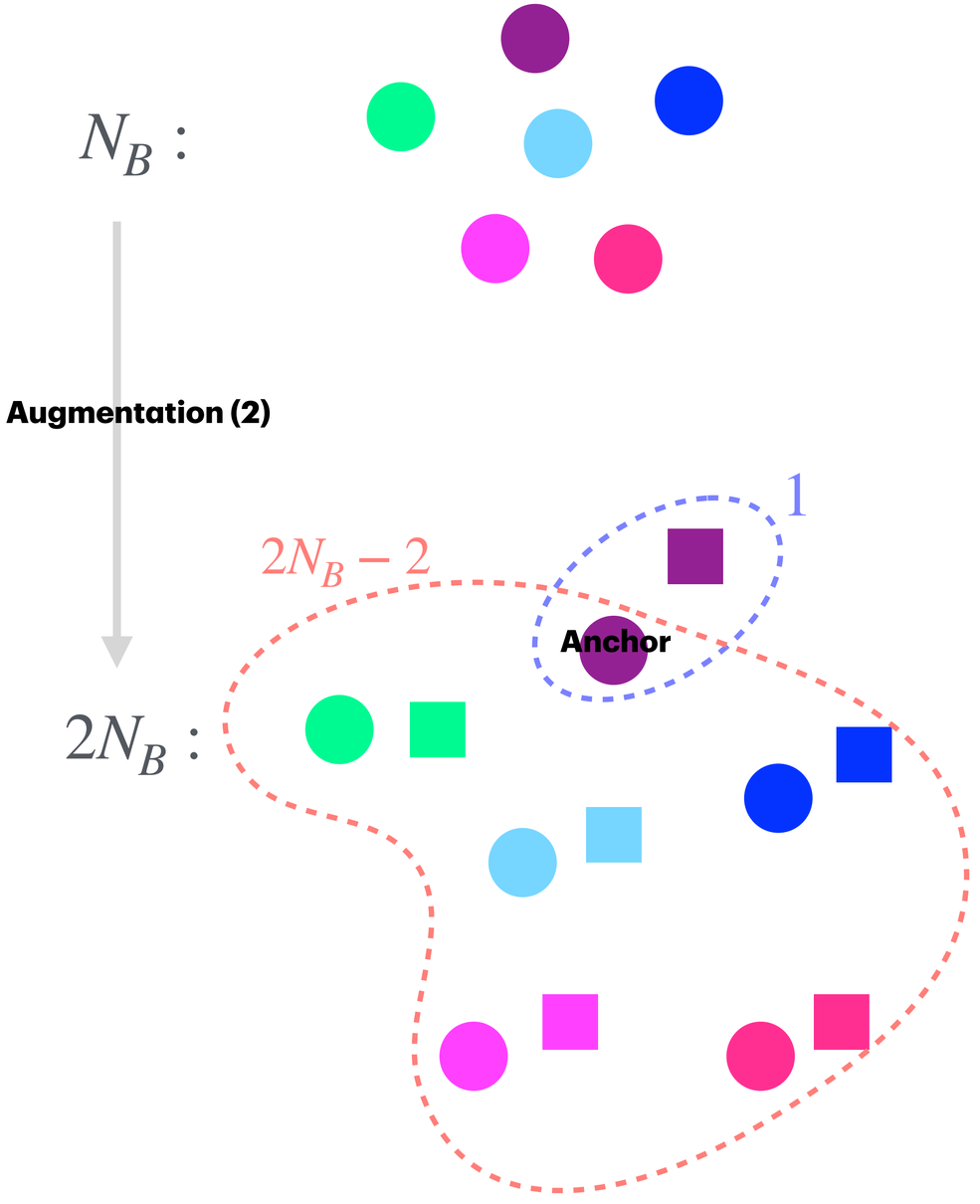
(iii) Contrastive Learning: How typically batches are being handled

The most Common Batch composition is:

Batch size: N_B original samples
Views per sample: 2 (typically)
Effective batch size: $2N_B$
Positive pairs: 1 per sample (the two views of the same image)
Negative pairs: $2(N_B - 2)$ per sample (all other views in the batch)

Every example is an anchor and compute average loss for ALL of them

Example:
If N_B=6:
6 original images → 12 views total
For $View_i$:
1 positive pair (its paired view)
1 0= negative pairs (all other views)



(iii) Contrastive Learning: Some domain agnostic augmentations based on data structure

Data modality	Example augmentations / views	What invariance it encourages
Sequences	Masking tokens/time points; cropping subsequences; local shuffling; small temporal shift; noise injection	Robustness to missing positions, local timing changes, and small sequence noise
Grids	Crop; resize; flip; rotation; blur; noise; color/intensity jitter; masking patches	Robustness to location, scale, lighting/intensity, and local corruption
Graphs	Edge dropout; node dropout; node-feature masking; subgraph sampling; random walks; graph diffusion views	Robustness to missing edges/nodes and local graph perturbations
Sets	Element dropout; random subsampling; feature noise; permutation; local jitter; set splitting	sampling variation, and missing elements, Robustness to element order (if needed)
Tabular	Feature masking; feature noise; dropout; mixup-like interpolation; corruption/reconstruction views	Robustness to missing/noisy features and measurement variation

IEEE Transactions on Knowledge and Data Engineering

A Comprehensive Survey on Data Augmentation

Jan. 2026, pp. 47-66, vol. 38
DOI Bookmark: 10.1109/TKDE.2025.3622600

(iii) Contrastive Learning:

Practical consequence of contrastive loss:

- Each anchor needs many negatives. In SimCLR, negatives come from the batch:

  larger batch → more negatives → stronger contrastive signal

  But this makes training expensive and batch-size sensitive.

- This raised a key question: Do we really need explicit negative pairs? BYOL showed that useful representations can be learned without negatives, using two networks, a predictor, and stop-gradient.

A Simple Framework for Contrastive Learning of Visual Representations

Ting Chen [1] Simon Kornblith [1] Mohammad Norouzi [1] Geoffrey Hinton [1]

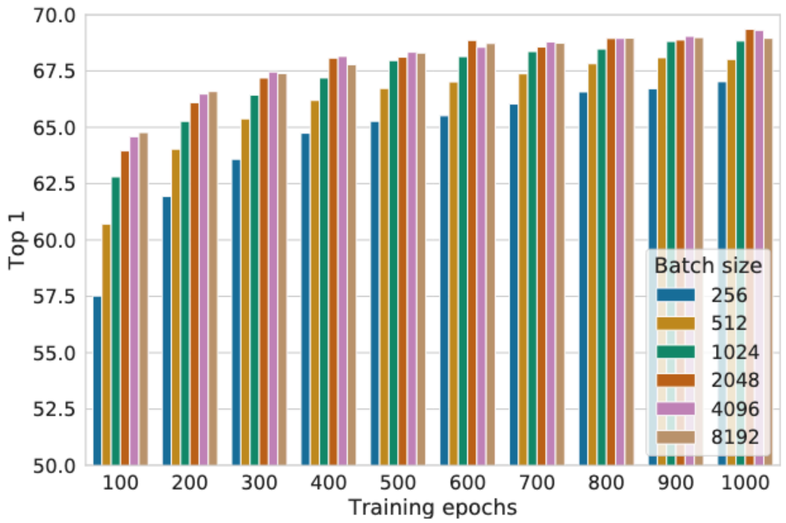
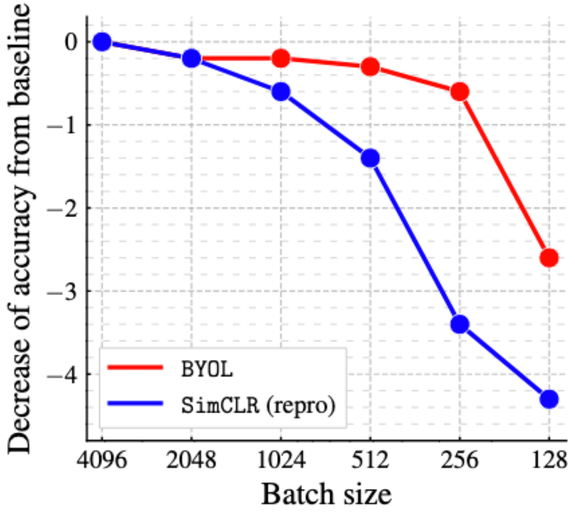


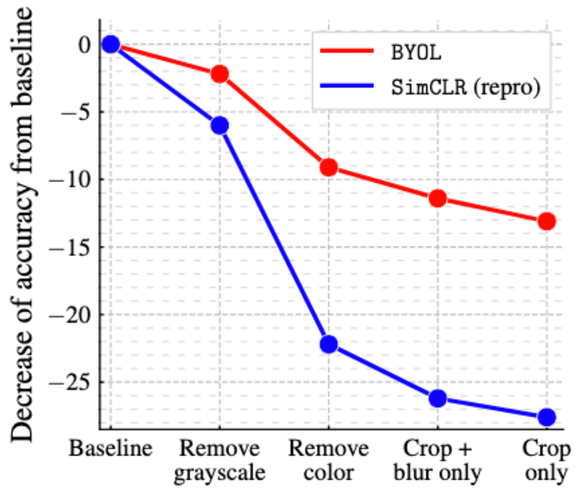
Figure 9. Linear evaluation models (ResNet-50) trained with different batch size and epochs. Each bar is a single run from scratch.[10]

Bootstrap Your Own Latent
A New Approach to Self-Supervised Learning

Jean-Bastien Grill[*,1], Florian Strub[*,1], Florent Altché[*,1], Corentin Tallec[*,1], Pierre H. Richemond[*,1,2]
Elena Buchatskaya[1], Carl Doersch[1], Bernardo Avila Pires[1], Zhaohan Daniel Guo[1]
Mohammad Gheshlaghi Azar[1], Bilal Piot[1], Koray Kavukcuoglu[1], Rémi Munos[1], Michal Valko[1]



(a) Impact of batch size

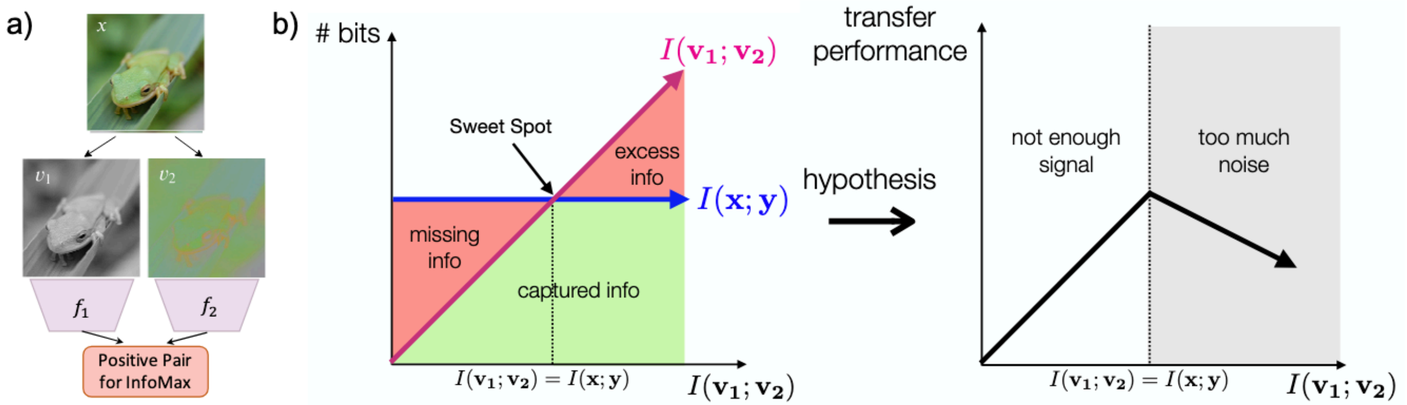


(iii) Contrastive Learning: What augmentations are good for learning?

Good augmentations should:

1. preserve task-relevant information
2. Remove nuisance variation
3. Avoid making positive pairs too easy or too different This is where domain knowledge matters: we must know which transformations preserve meaning in the data.
4. Some approaches learn the transformation as well (Cubuk, Ekin D., et al. "Autoaugment: Learning augmentation policies from data." *arXiv preprint arXiv:1805.09501* (2018))

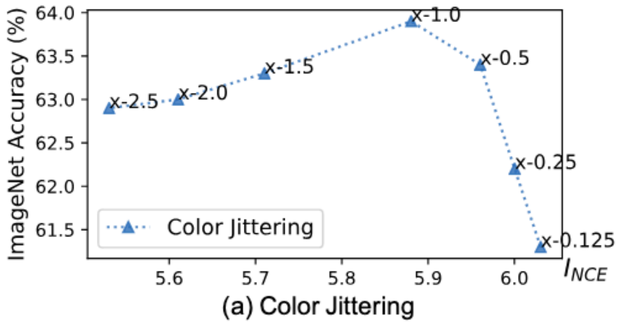
5. good views share the minimal information needed for the downstream task.

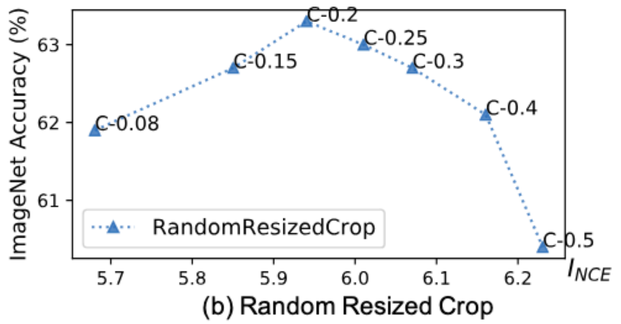


What Makes for Good Views for Contrastive Learning?

Yonglong Tian MIT CSAIL
Chen Sun Google, Brown University
Ben Poole Google Research
Dilip Krishnan Google Research
Cordelia Schmid Google Research
Phillip Isola MIT CSAIL

Tian, Yonglong, et al. "What makes for good views for contrastive learning?." *Advances in neural information processing systems* 33 (2020): 6827-6839.





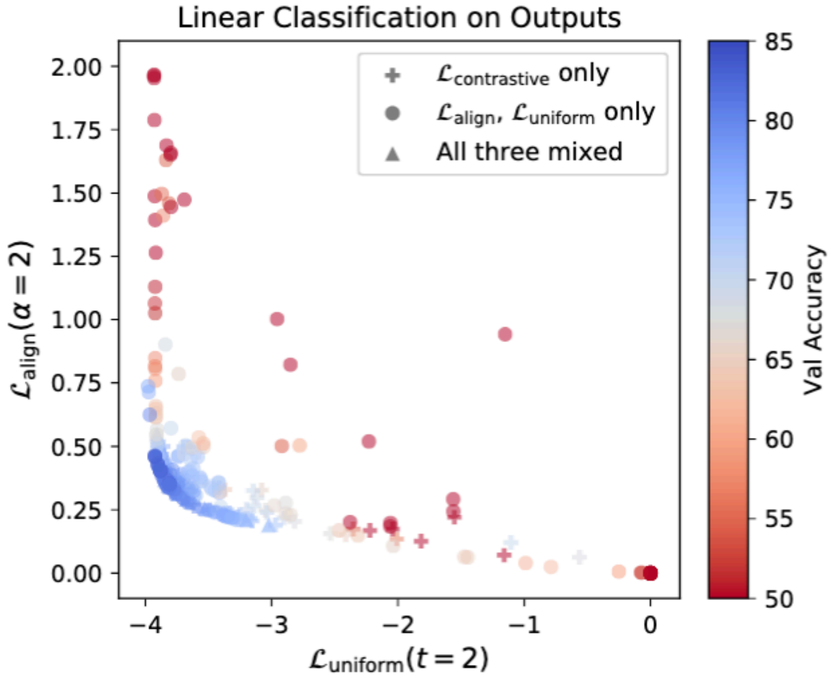
(iii) Contrastive Learning: What does the contrastive learning actually do?

Let's rewrite the contrastive loss in the limit of infinite number of samples:

For simplicity we define: $u = z_i^{(a)}$ $\quad v = z_i^{(b)}$ $\quad z_n^- =$ negative examples

The loss is: $\mathscr{L}_{cont} = -\log \frac{e^{sim(u,v)/\tau}}{e^{sim(u,v)/\tau} + \sum_{n=1}^{N_{neg}} e^{sim(u,z_n^-)/\tau}}$

Now separate numerator and denominator: $\mathscr{L}_{cont} = -\frac{sim(u,v)}{\tau} + \log N_{neg} + \log\left[\frac{1}{N_{neg}} e^{sim(u,v)/\tau} + \frac{1}{N_{neg}} \sum_{n=1}^{N_{neg}} e^{sim(u,z_n^-)/\tau}\right]$



$\mathscr{L}_{alighn}$

Alignment:

Minimize when

(similar) point are nearby

Const

~0 if

$N_{neg} \to \inf$

$\mathscr{L}_{uniform}$

Uniformity:
Minimize when (dissimilar) points are far away uniformly

Wang, Tongzhou, and Phillip Isola. "Understanding contrastive representation learning through alignment and uniformity on the hypersphere." *International conference on machine learning*. PMLR, 2020.

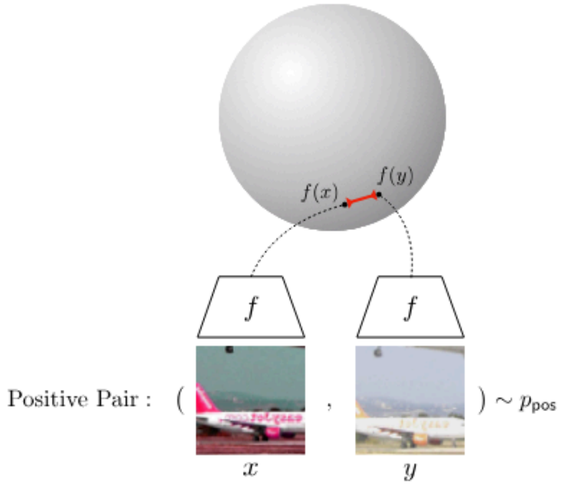
(iii) Contrastive Learning: What does the contrastive learning actually do?

Contrastive learning shapes the representation space by:

1. Alignment: views of the same input are pulled together

2. Separation : views from different inputs are pushed apart

3. Robustness: irrelevant augmentations are ignored, while shared semantic information is preserved



Alignment: Similar samples have similar features.
(Figure inspired by Tian et al. (2019).)

Uniformity: Preserve maximal information.

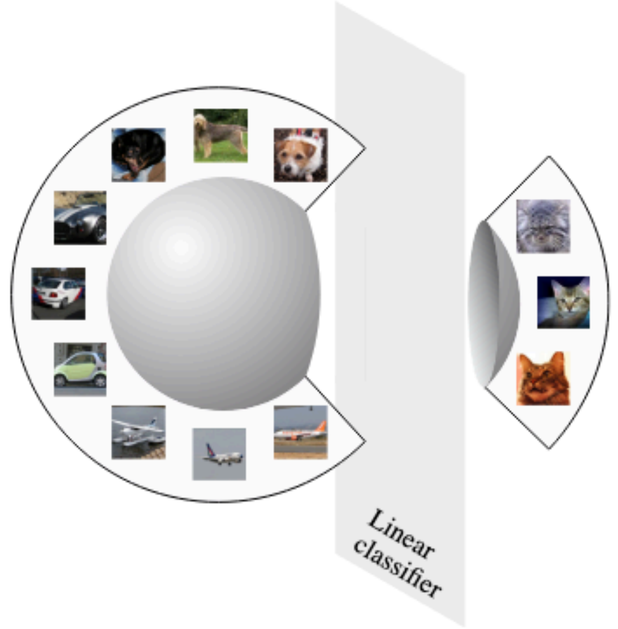


Figure 2: **Hypersphere:** When classes are well-clustered (forming spherical caps), they are linearly separable. The same does not hold for Euclidean spaces.

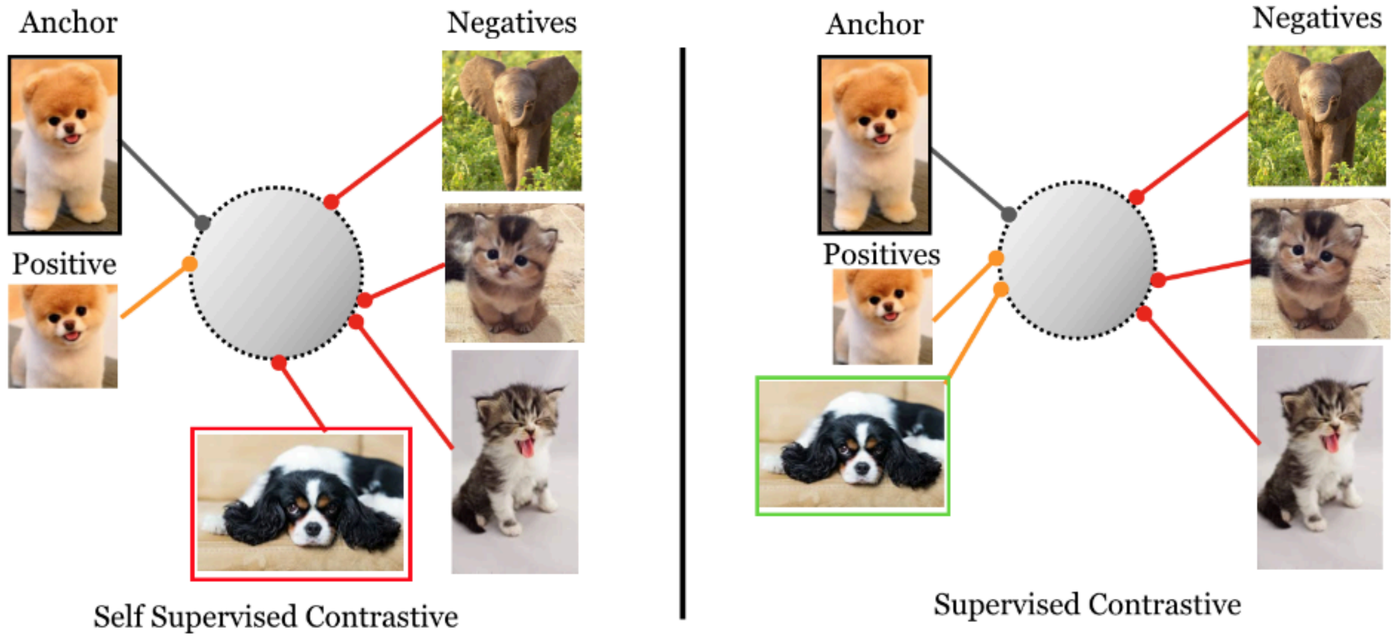
Understanding Contrastive Representation Learning through Alignment and Uniformity on the Hypersphere

Tongzhou Wang [1] **Phillip Isola** [1]

Wang, Tongzhou, and Phillip Isola. "Understanding contrastive representation learning through alignment and uniformity on the hypersphere." *International conference on machine learning*. PMLR, 2020.

(iii) Contrastive Learning: Also semi-supervised CL methods exist

In these methods we use the labeled data to define the positive samples based on the labeled data (if exists as partly).



Supervised Contrastive Learning

Prannay Khosla *
Google Research

Piotr Teterwak * †
Boston University

Chen Wang †
Snap Inc.

Aaron Sarna ‡
Google Research

Yonglong Tian †
MIT

Phillip Isola †
MIT

Aaron Maschinot
Google Research

Ce Liu
Google Research

Dilip Krishnan
Google Research

Khosla, Prannay, et al. "Supervised contrastive learning." *Advances in neural information processing systems* 33 (2020): 18661-18673.

Recipe Card: **Contrastive Learning**

Type of Learning: **Self-supervised**
Data type: **any unlabeled data**

Data

1- Load libraries: Basics + PyTorch

2-Load the Dataset: $\mathcal{D} = \{x_i\}_{i=1}^{N}$

3-Define augmentation functions:

$$T_a \qquad ; a = 1, \ldots, V_{total}$$

4-For each input x_i , create multiple views:

$$x_i^{(a)} = T_a(x_i), a = 1, \ldots, V_{total}$$

5-Define positives and negatives:

-positives are views of the same input:

$$x_i^{(a)}, x_i^{(b)},\ b \neq a$$

-Negatives are views from different inputs:

$$x_i^{(a)}, x_j^{(c)},\ j \neq i$$

NN model

6- Build an encoder: $h_i^{(a)} = f_\theta(x_i^{(a)})$

-Here h is the representation we want to keep

7-Add a projection head: $z_i^{(a)} = g_\phi(h_i^{(a)})$

-The contrastive loss is applied to z, not directly to h.

- Projection head is usually a linear layer or a small MLP

-All together the architecture is:

$$x_i^{(a)} \to f_\theta \to h_i^{(a)} \to g_\phi \to z_i^{(a)}$$

8- Define similarity, usually cosine similarity:

$$\text{sim}(u, v) = \frac{u^\top v}{\|u\|\|v\|},\ u, v \in Z$$

Training

9-Define contrastive DataLoader / batch structure

- We define the metric in the projection output: $z = g_\phi(h) = g_\phi(f_\theta(x))$
- Usually only V_B (~2-6) number of augmentations is applied (and not V_{total})
- Per anchor we have $N_{positive} \approx 1$ positive and $N_{neg} = V_{total}N_B - V_{total}$ negative pairs

10- Define positives and negatives inside the batch

-For anchor $z_i^{(a)}$, positive $z_i^{(b)}$, and negatives: $z_1^-, z_2^-, \ldots, z^-{}_{N_{neg}}$

11- Define the contrastive loss: $\mathcal{L}_i^{(a)} = -\log \frac{e^{sim\left(z_i^{(a)}, z_i^{(b)}\right)/\tau}}{e^{sim\left(z_i^{(a)}, z_i^{(b)}\right)/\tau} + \sum_{n=1}^{N\text{neg}} e^{sim\left(z_i^{(a)}, z_n^-\right)/\tau}}$

- Total loss of the batch for all anchors: $\mathcal{L}_{total,cont} = \frac{1}{N_B V_B} \sum_{i=1}^{N_B} \sum_{a=1}^{V_B} \mathcal{L}_i^{(a)}$

12- Train the model

- For each batch: $x_i^{(a)} \to f_\theta \to h_i^{(a)} \to g_\phi \to z_i^{(a)}$
- Then: normalize $z \to$ compute pairwise similarities $\to$ identify positives using batch indices $\to$ treat all other samples as negatives$\to$ compute contrastive loss $\to$ update f_θ and g_ϕ

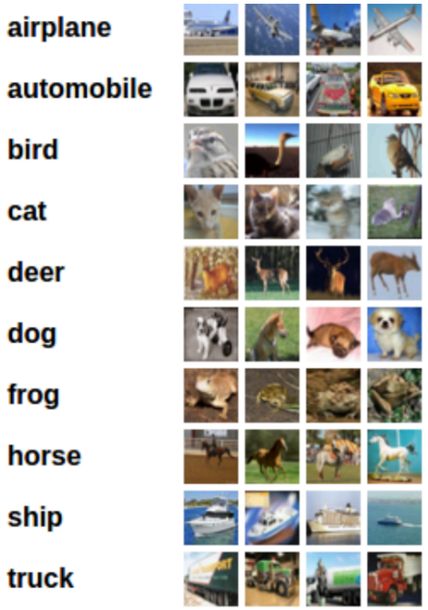
13- After training:

- Discard the projection head, g_ϕ
- Keep the encoder: $h = f_\theta(x)$
- Use h for downstream tasks, e.g linear probe / kNN / fine-tuning / transfer

(iii) Contrastive Learning: Example setup

Dataset → **Augmentation** → **Model:** Encoder+Head → **Training** → **Analysis**

CIFAR-10



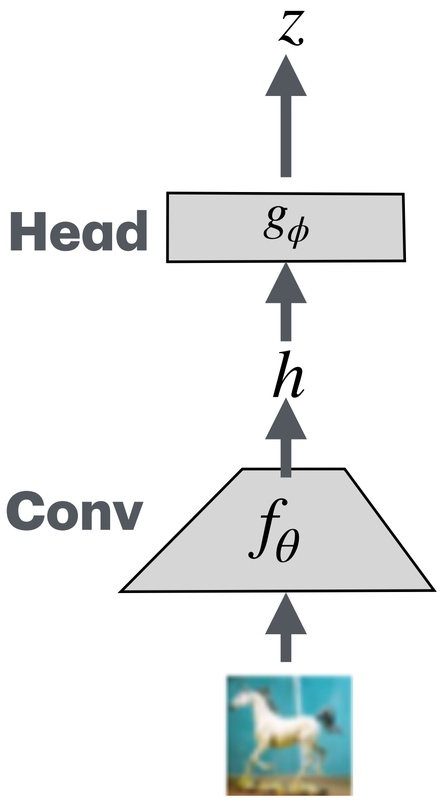
Random crop

Flip

Color jitter

Grayscale

Blur/noise



Contrastive

B=256 (small)

Visualize h

Classification via h

(iii) Contrastive Learning: **Example setup**

Mapped point of CIFAR-10 data points in the Z space:

Dimension of $dim_Z = 3$ (Hypersphere)

Representation is learned via self-supervised learning