CENG501 – Deep Learning

Week 14

Spring 2026

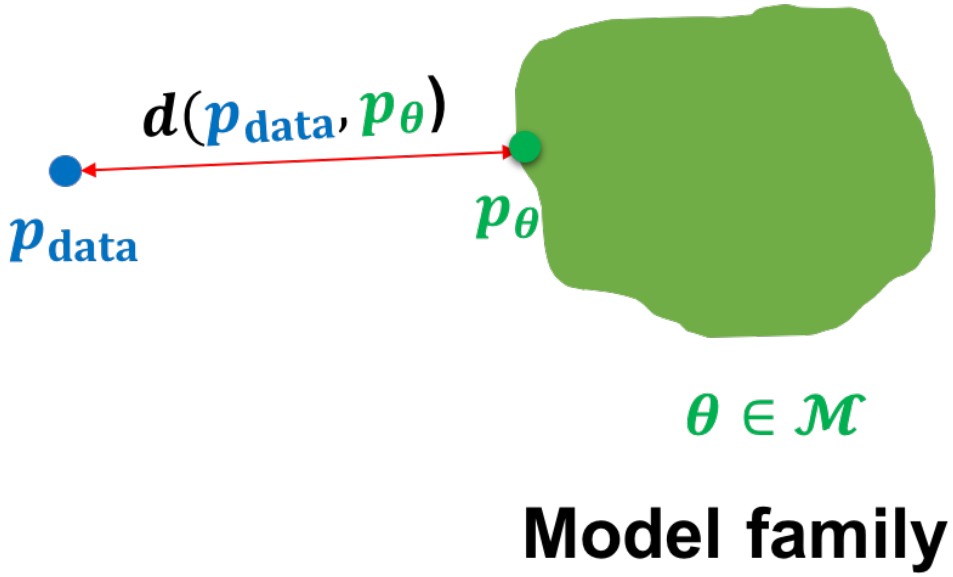
Sinan Kalkan

Dept. of Computer Engineering, METU

Previously on CSCENG501

Generative Modeling

- Learning the probability distribution of data



$p_\theta \equiv p_{\text{model}}$

Figure: https://deepgenerativemodels.github.io/notes/introduction/

Previously on CENG501

Taxonomy of Generative Models

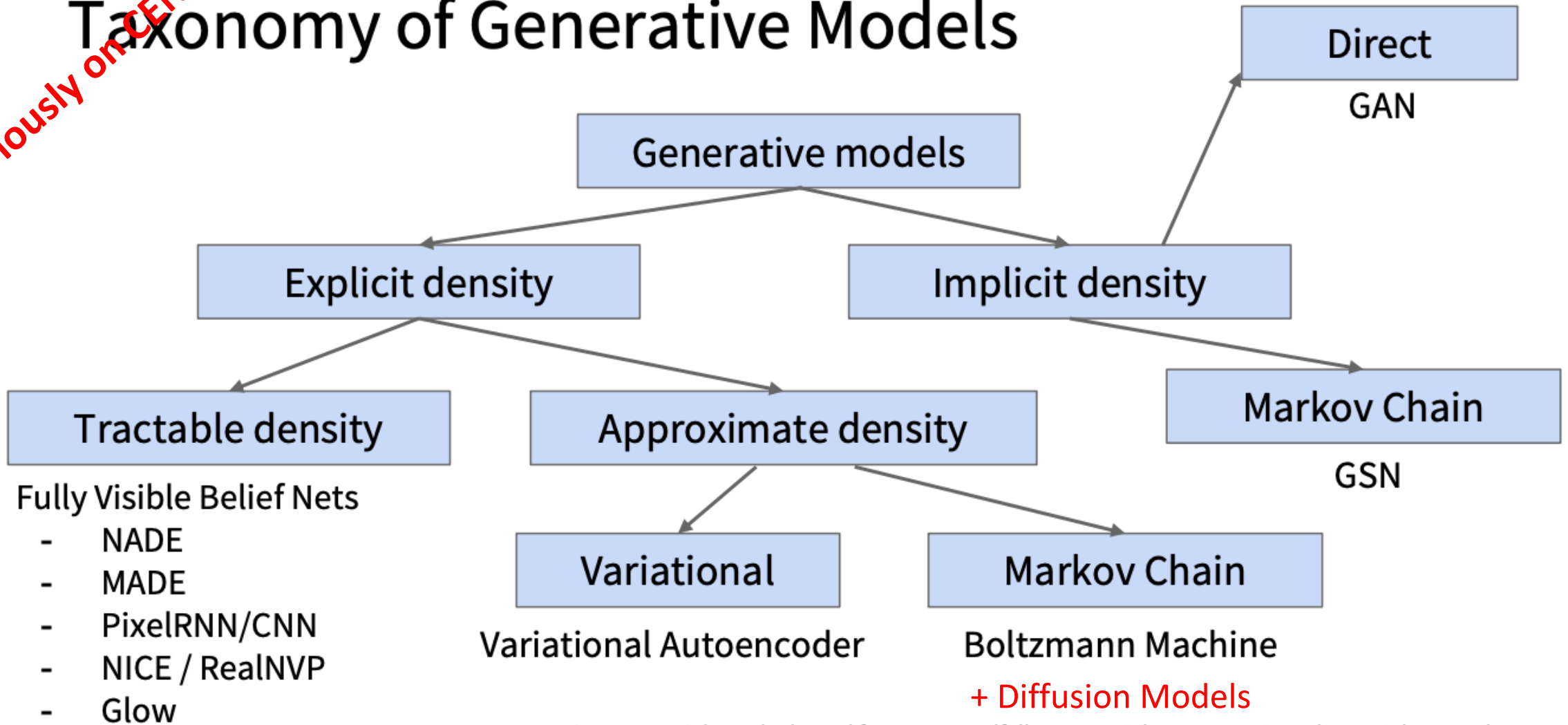
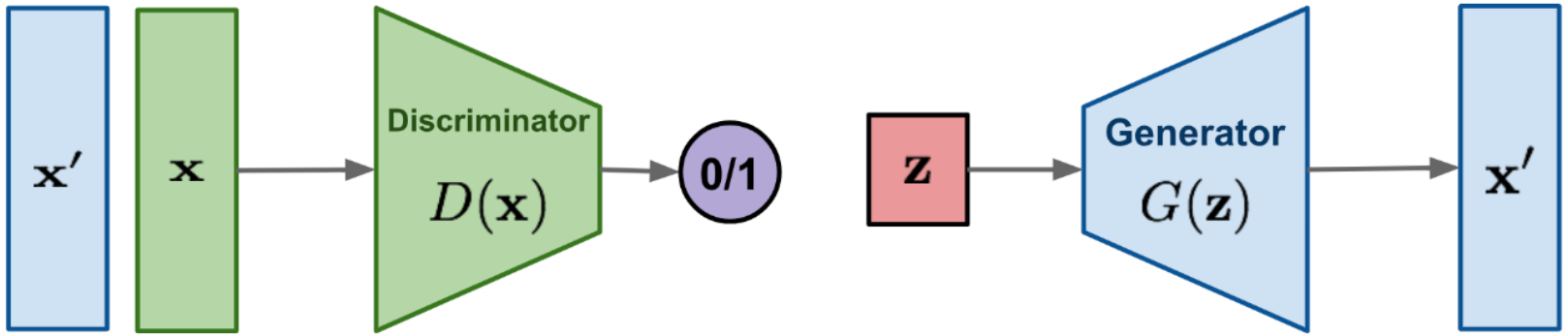


Figure copyright and adapted from Ian Goodfellow, Tutorial on Generative Adversarial Networks, 2017.

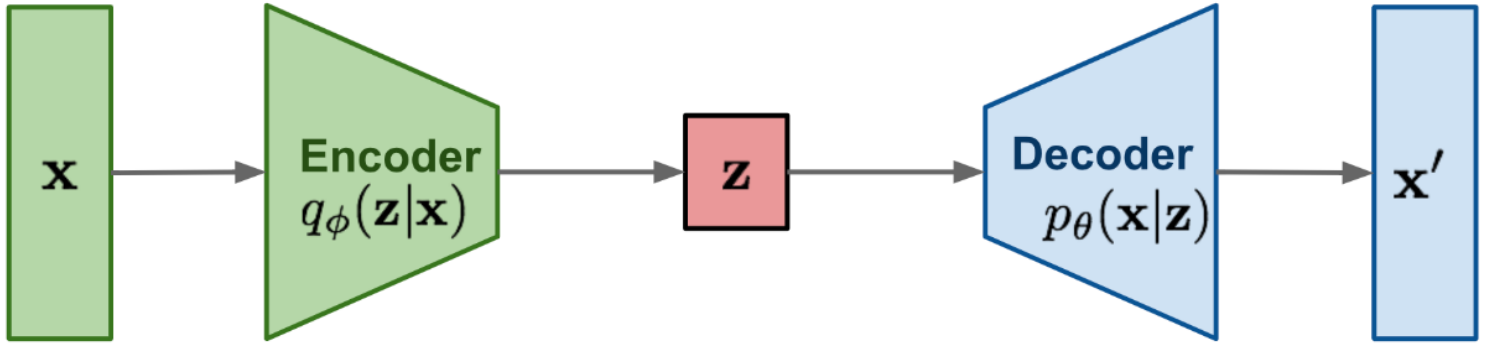
Slide: https://cs231n.stanford.edu/slides/2024/lecture_13.pdf

Previously on CENG501

GAN: minimax the classification error loss.



VAE: maximize ELBO.



Flow-based generative models: minimize the negative log-likelihood

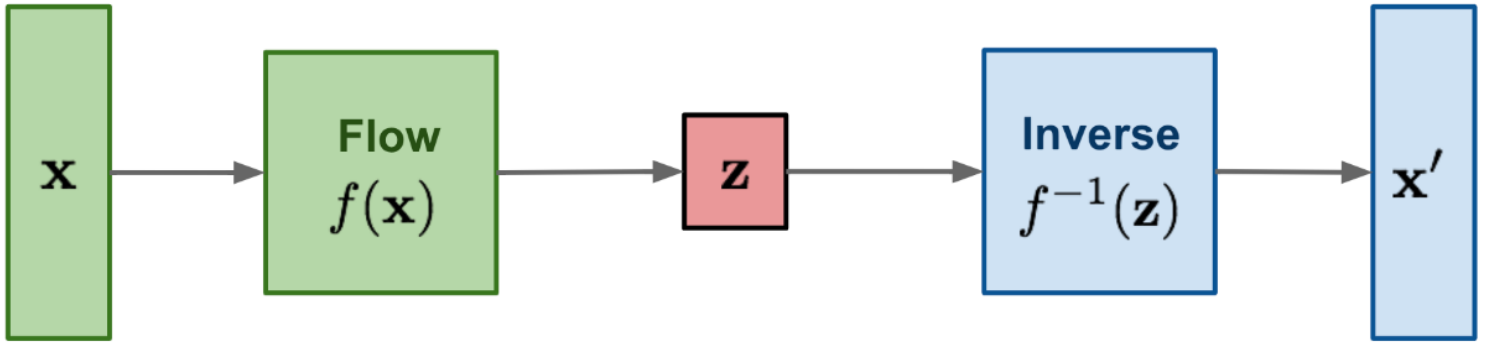


Figure: https://lilianweng.github.io/posts/2018-10-13-flow-models/

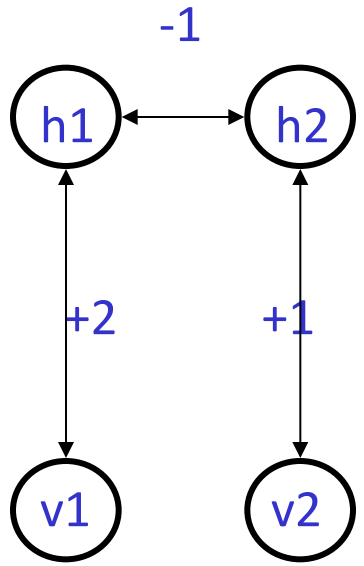
Boltzmann Machines: An Example

Previously on CENG501

$$E(\mathbf{s}) = -\sum_i \sum_{j<i} w_{ij} s_i s_j + \sum_i \theta_i s_i$$

v	h	$-E$	e^{-E}	$p(\mathbf{v},\mathbf{h})$	$p(\mathbf{v})$
1 1	1 1	2	7.39	.186	0.466
1 1	1 0	2	7.39	.186	
1 1	0 1	1	2.72	.069	
1 1	0 0	0	1	.025	
1 0	1 1	1	2.72	.069	0.305
1 0	1 0	2	7.39	.186	
1 0	0 1	0	1	.025	
1 0	0 0	0	1	.025	
0 1	1 1	0	1	.025	0.144
0 1	1 0	0	1	.025	
0 1	0 1	1	2.72	.069	
0 1	0 0	0	1	.025	
0 0	1 1	-1	0.37	.009	0.084
0 0	1 0	0	1	.025	
0 0	0 1	0	1	.025	
0 0	0 0	0	1	.025	

total = 39.70

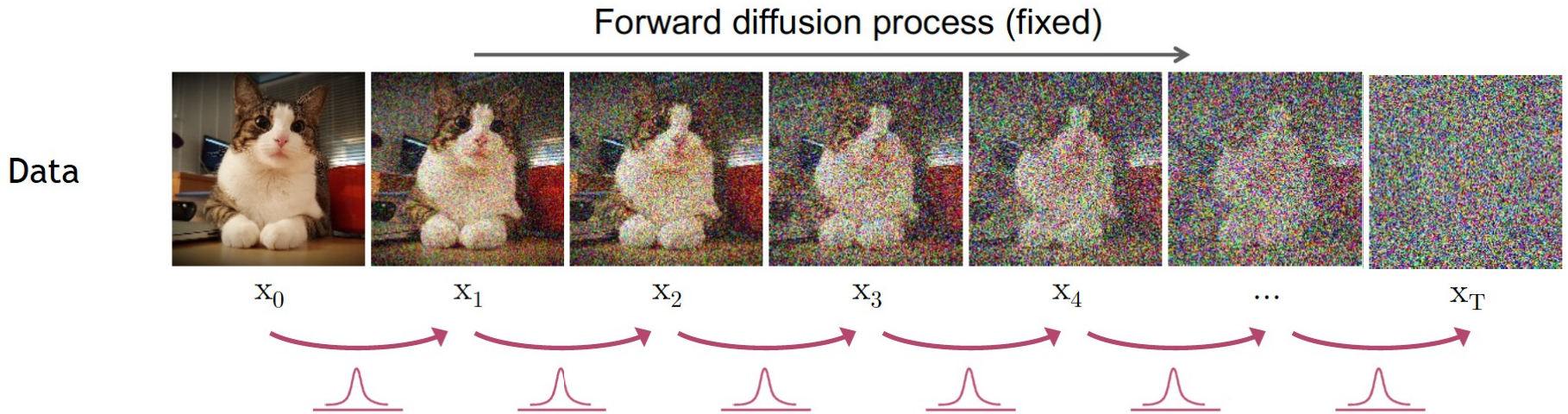


Adapted from G. Hinton

Previously on CENG501

Forward Diffusion Process

The formal definition of the forward process in T steps:

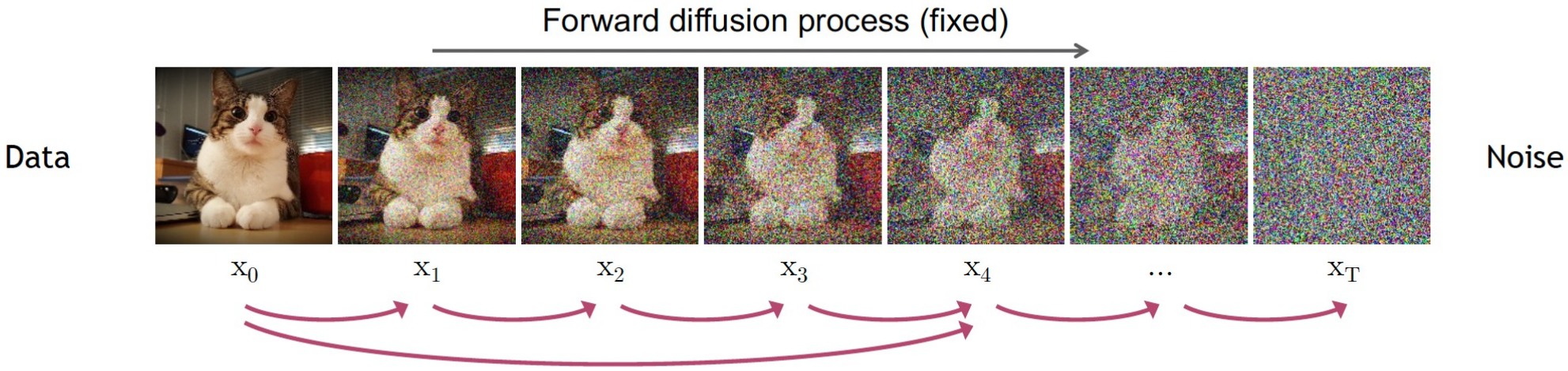


$$q(\mathbf{x}_t|\mathbf{x}_{t-1}) = \mathcal{N}(\mathbf{x_t}; \sqrt{1-\beta_t}\mathbf{x_{t-1}}, \beta_t\mathbf{I}) \quad \Rightarrow \quad q(\mathbf{x}_{1:T}|\mathbf{x}_0) = \prod_{t=1}^{T} q(\mathbf{x}_t|\mathbf{x}_{t-1}) \qquad \text{(joint)}$$

Slide: https://cvpr2022-tutorial-diffusion-models.github.io/

Previously on CENG501

Diffusion Kernel



Define $\bar{\alpha}_t = \prod_{s=1}^{t}(1-\beta_s)$ ➡ $q(\mathbf{x}_t|\mathbf{x}_0) = \mathcal{N}(\mathbf{x}_t; \sqrt{\bar{\alpha}_t}\mathbf{x}_0, (1-\bar{\alpha}_t)\mathbf{I}))$ (Diffusion Kernel)

For sampling: $\mathbf{x}_t = \sqrt{\bar{\alpha}_t}\,\mathbf{x}_0 + \sqrt{(1-\bar{\alpha}_t)}\,\epsilon$ where $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$

β_t values schedule (i.e., the noise schedule) is designed such that $\bar{\alpha}_T \to 0$ and $q(\mathbf{x}_T|\mathbf{x}_0) \approx \mathcal{N}(\mathbf{x}_T; \mathbf{0}, \mathbf{I}))$

Slide: https://cvpr2022-tutorial-diffusion-models.github.io/

Today

- Self-Supervised Learning
- Reinforcement Learning
- Graph Neural Networks

Administrative Notes

- Project next steps:
  - Milestones:
    1. Milestone (April 10, midnight):
       - Read & understand the paper
       - Download the datasets
       - Prepare the Readme file excluding the results & conclusion
    2. Milestone (May 4, midnight)
       - The results of the first experiment
    3. Milestone (June 1, midnight)
       - Final report (Readme file)
       - Repo with all code & trained models

Self-supervised learning

Motivation for SSL

- Labeling is expensive
- Approach:
  - Step 1: Formulate a supervision signal based on the unlabeled data.
  - Step 2: Train an encoder with this self-supervision signal.
  - Step 3: Use the encoder as feature extractor. Keep the encoder frozen or finetune it.

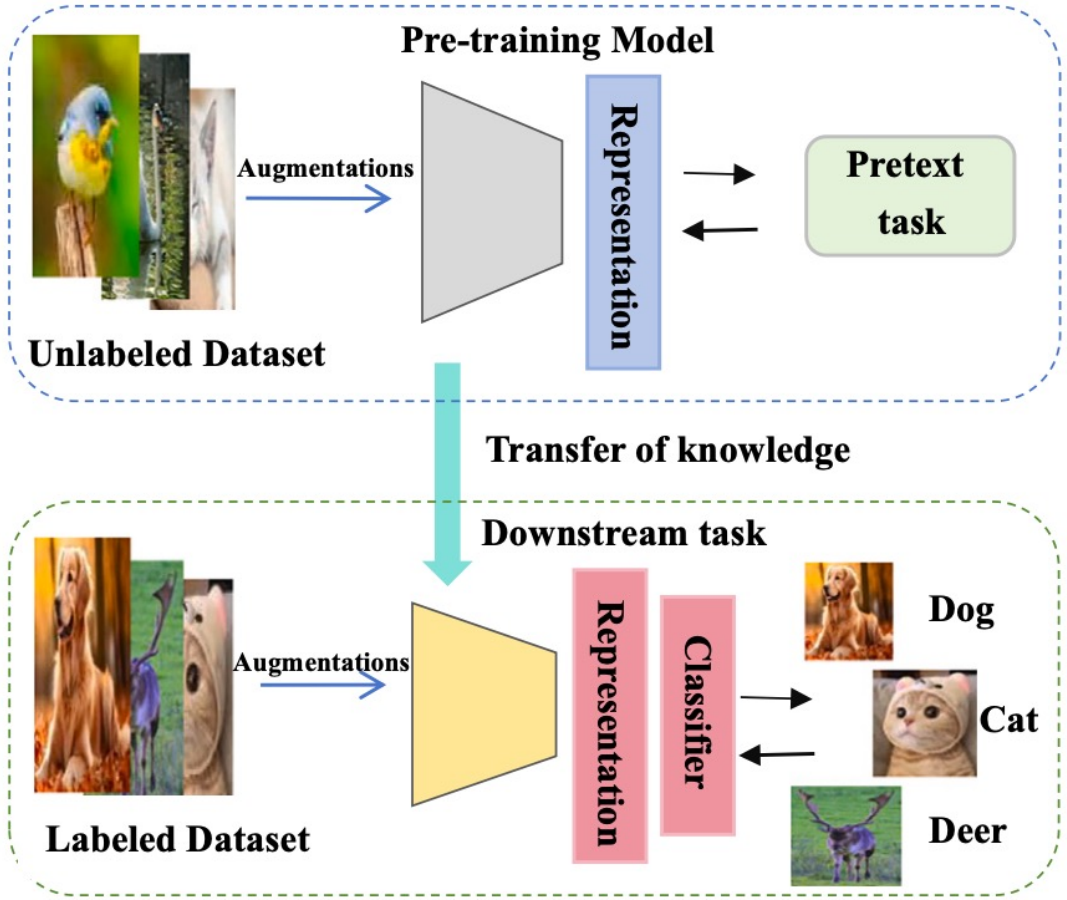


Fig: https://www.sciencedirect.com/science/article/pii/S0925231225020818

Pretext Tasks

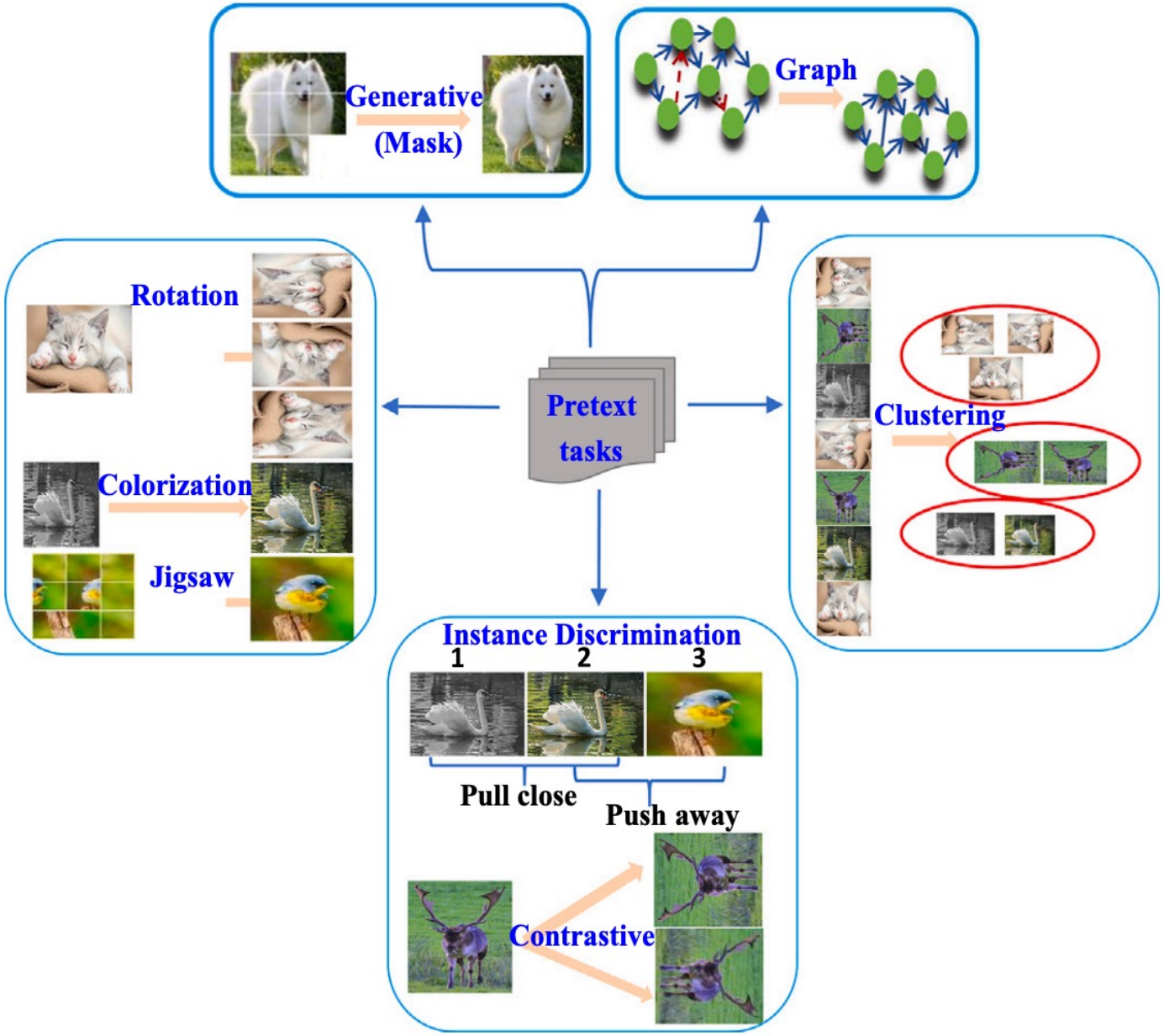
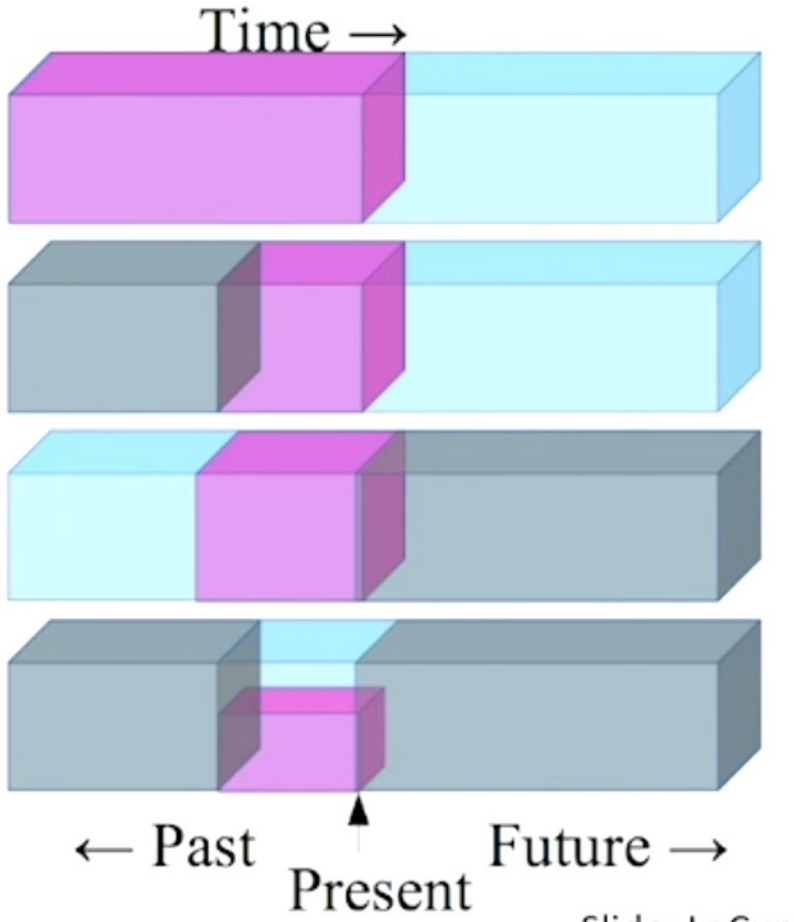


Fig: https://www.sciencedirect.com/science/article/pii/S0925231225020818

- Predict any part of the input from any other part.
- Predict the future from the past.
- Predict the future from the recent past.
- Predict the past from the present.
- Predict the top from the bottom.
- Predict the occluded from the visible
- Pretend there is a part of the input you don't know and predict that.



Slide: LeCun

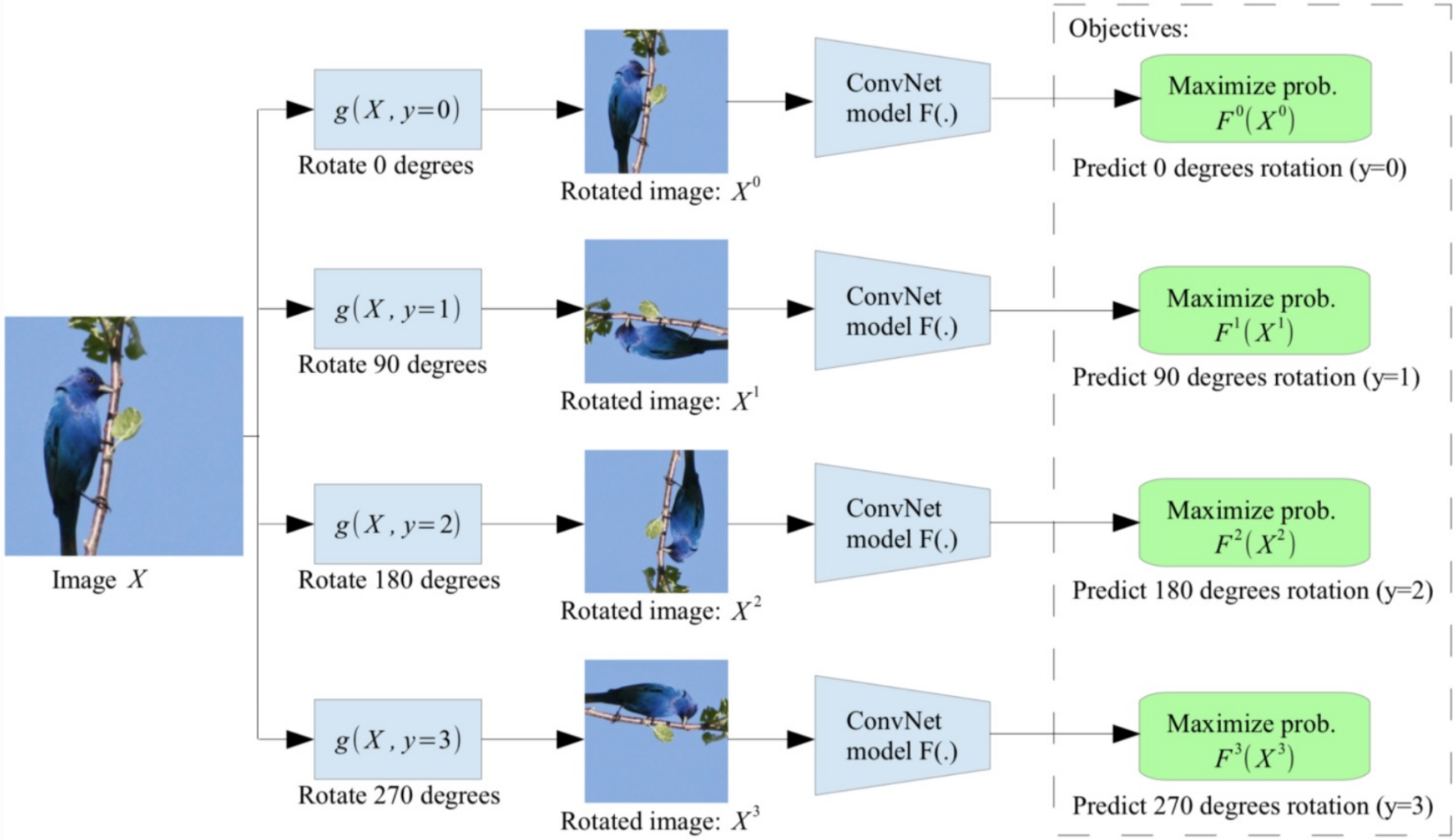


Fig. 3. Illustration of self-supervised learning by rotating the entire input images. The model learns to predict which rotation is applied. (Image source: Gidaris et al. 2018)

From: https://lilianweng.github.io/lil-log/2019/11/10/self-supervised-learning.html

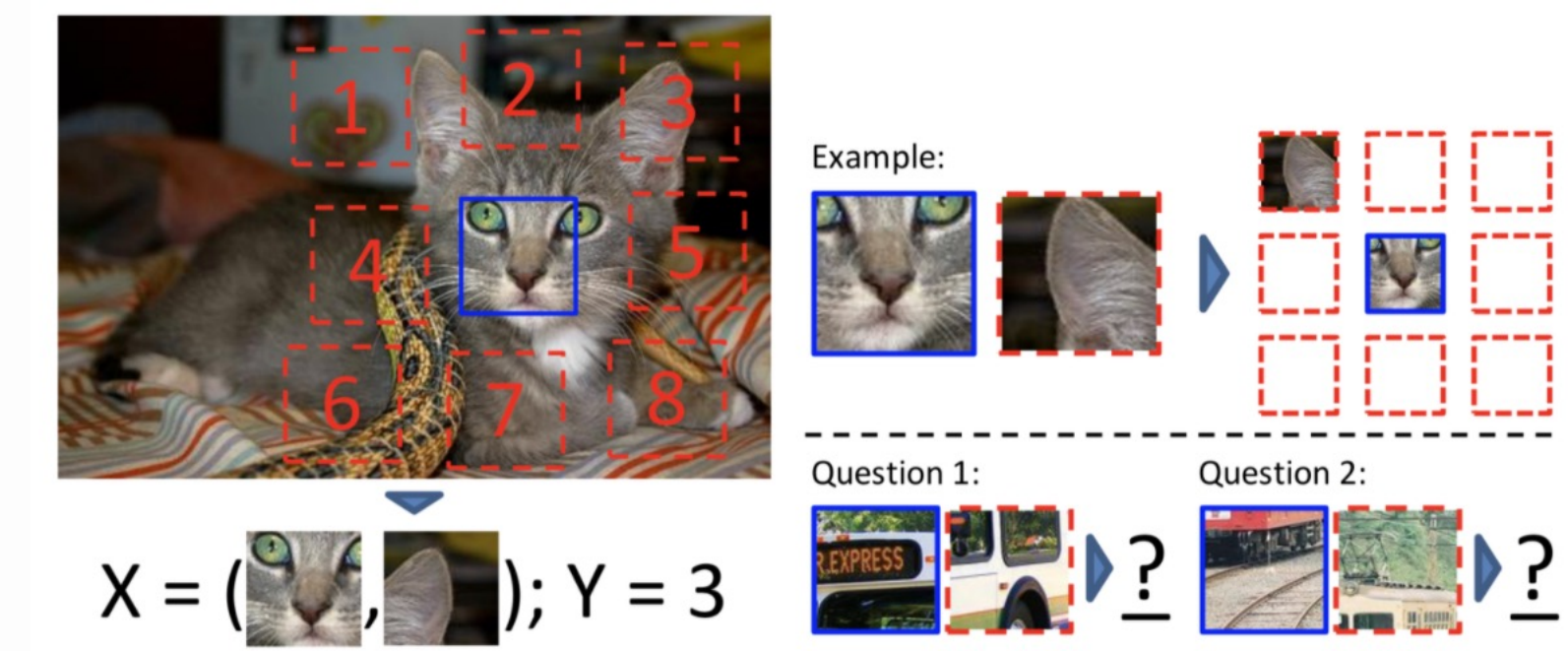


Fig. 4. Illustration of self-supervised learning by predicting the relative position of two random patches. (Image source: Doersch et al., 2015)

From: https://lilianweng.github.io/lil-log/2019/11/10/self-supervised-learning.html

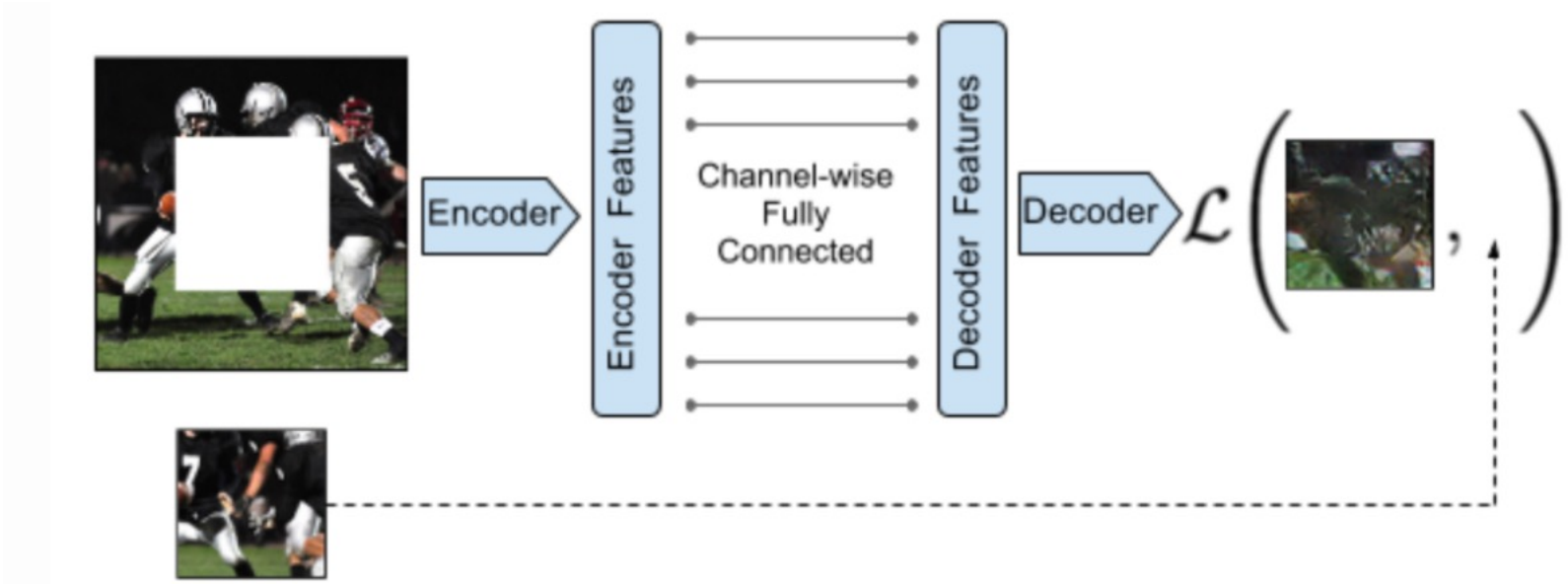
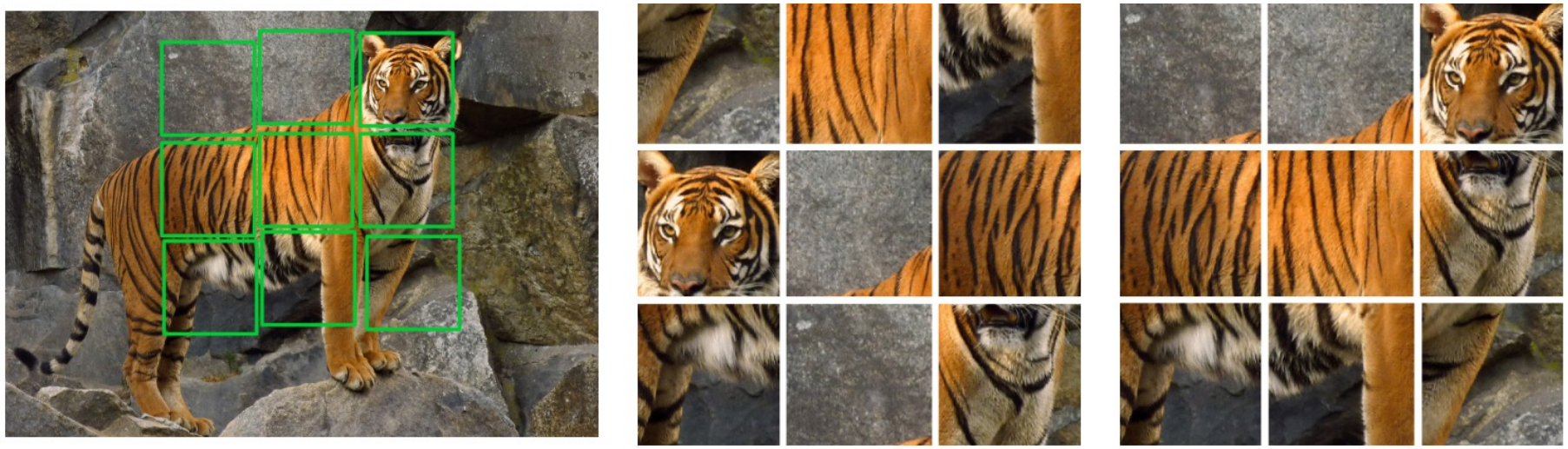


Fig. 8. Illustration of context encoder. (Image source: Pathak, et al., 2016)

From: https://lilianweng.github.io/lil-log/2019/11/10/self-supervised-learning.html

Noroozi & Favaro, Unsupervised Learning of Visual Representations by Solving Jigsaw Puzzles, 2017.



(a) (b) (c)

Fig. 1: Learning image representations by solving Jigsaw puzzles. (a) The image from which the tiles (marked with green lines) are extracted. (b) A puzzle obtained by shuffling the tiles. Some tiles might be directly identifiable as object parts, but others are ambiguous (*e.g.*, have similar patterns) and their identification is much more reliable when all tiles are jointly evaluated. In contrast, with reference to (c), determining the relative position between the central tile and the top two tiles from the left can be very challenging [10].

Zhang et al., Colorful Image Colorization, 2016.

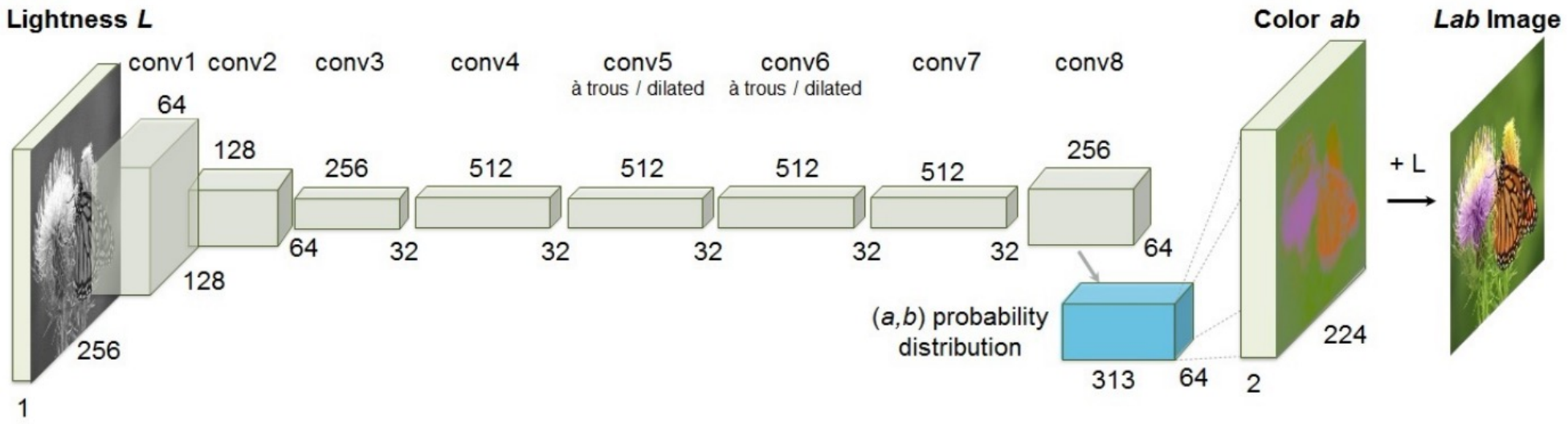
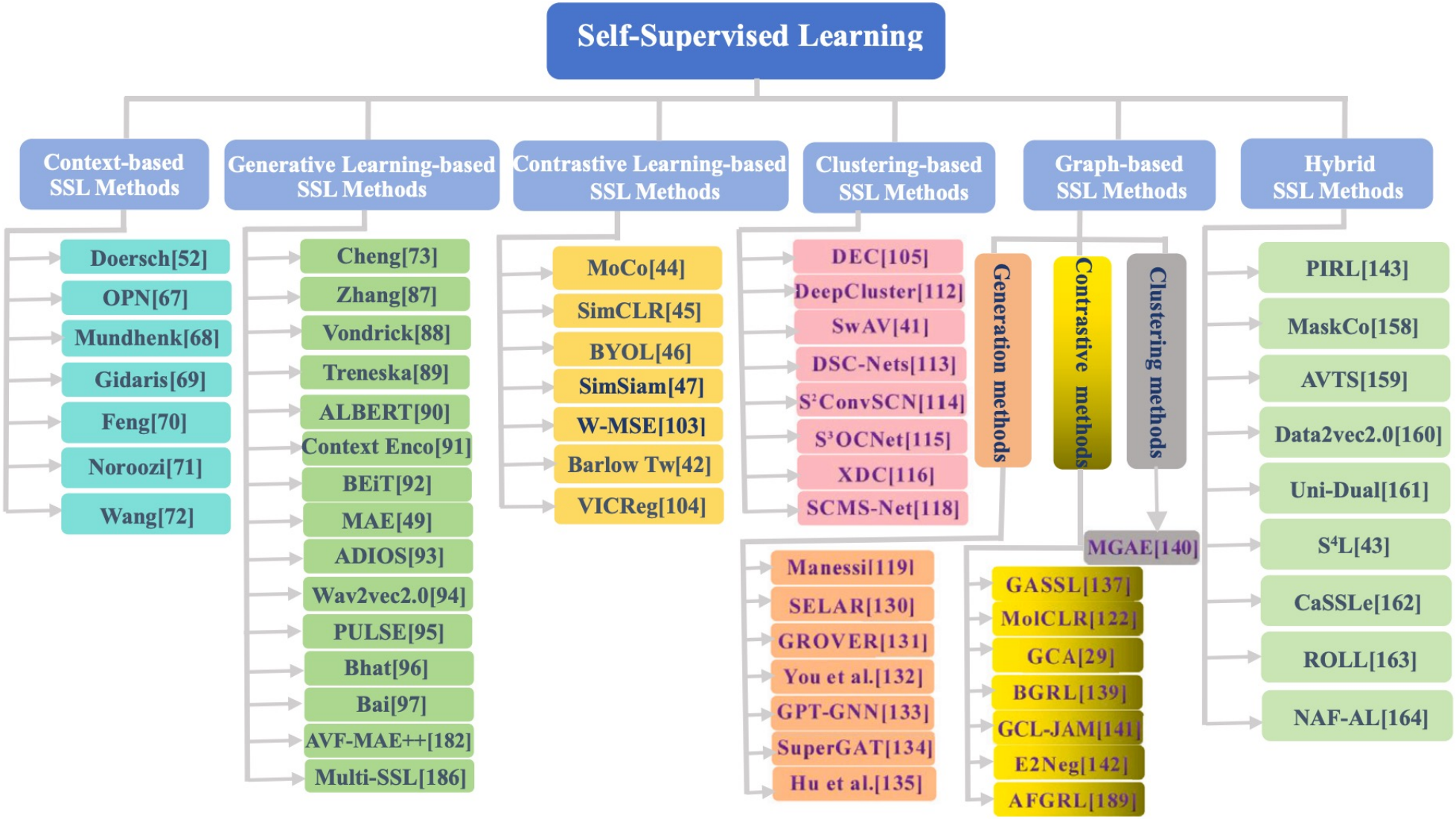


Fig. 2. Our network architecture. Each `conv` layer refers to a block of 2 or 3 repeated `conv` and `ReLU` layers, followed by a `BatchNorm` [30] layer. The net has no `pool` layers. All changes in resolution are achieved through spatial downsampling or upsampling between `conv` blocks.

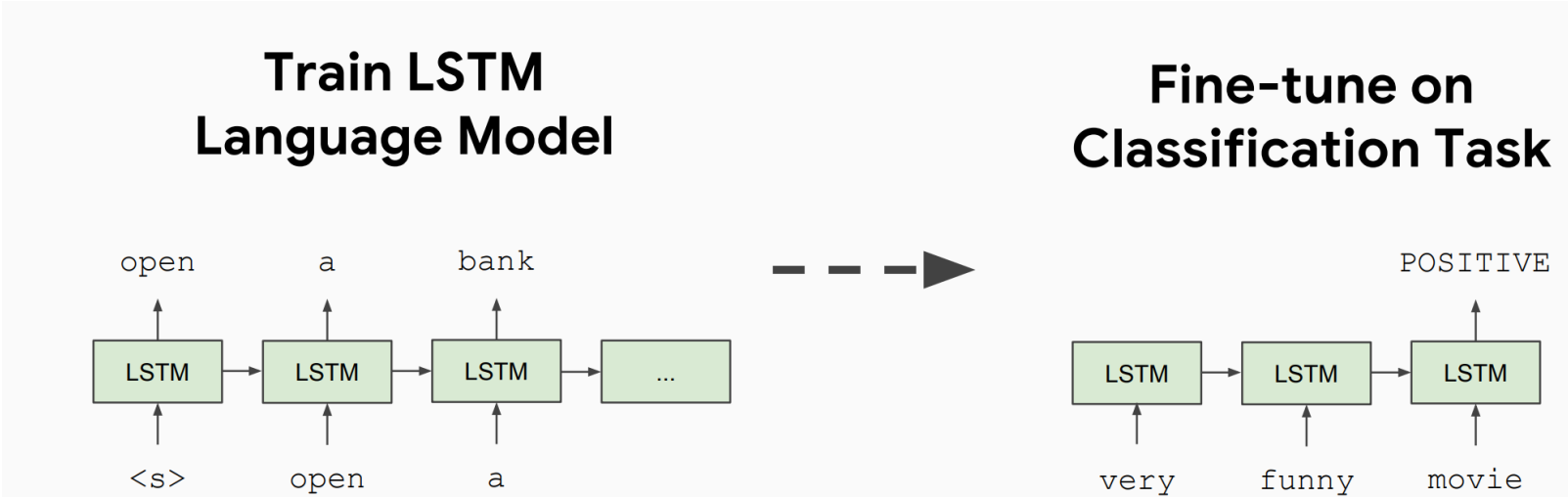


https://www.sciencedirect.com/science/article/pii/S0925231225020818

Generative Approaches

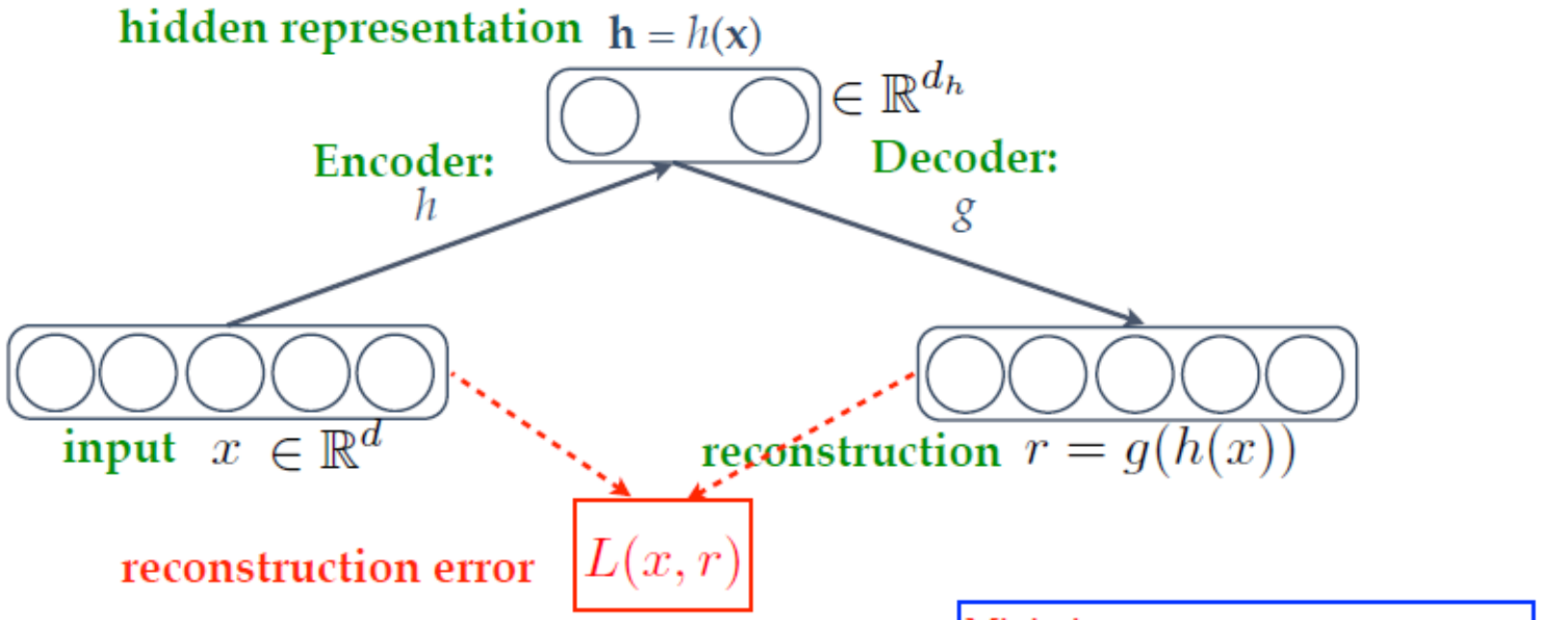
Autoregressive Approaches

$$\max_{\theta} p_\theta(\mathbf{x}) = \sum_{t=1}^{T} \log p_\theta\left(x_t \mid \mathbf{x}_{1:t-1}\right)$$



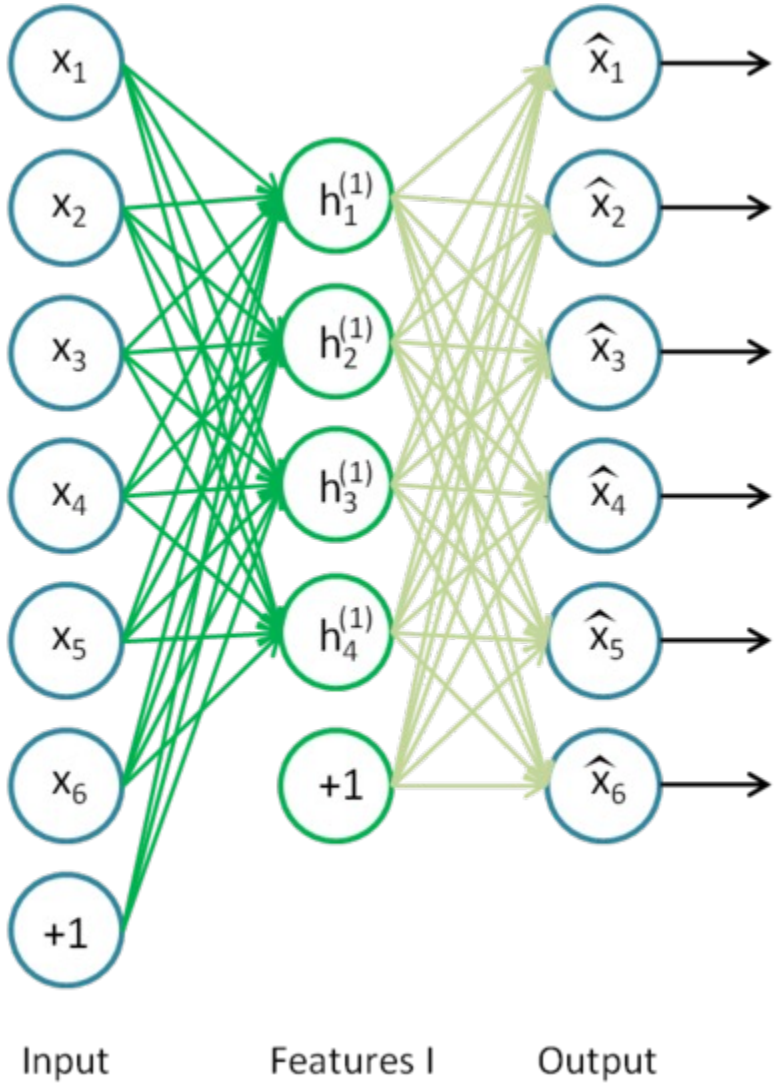
Slide: Jacob Devlin
https://nlp.stanford.edu/seminar/details/jdevlin.pdf

Auto-encoder Approaches



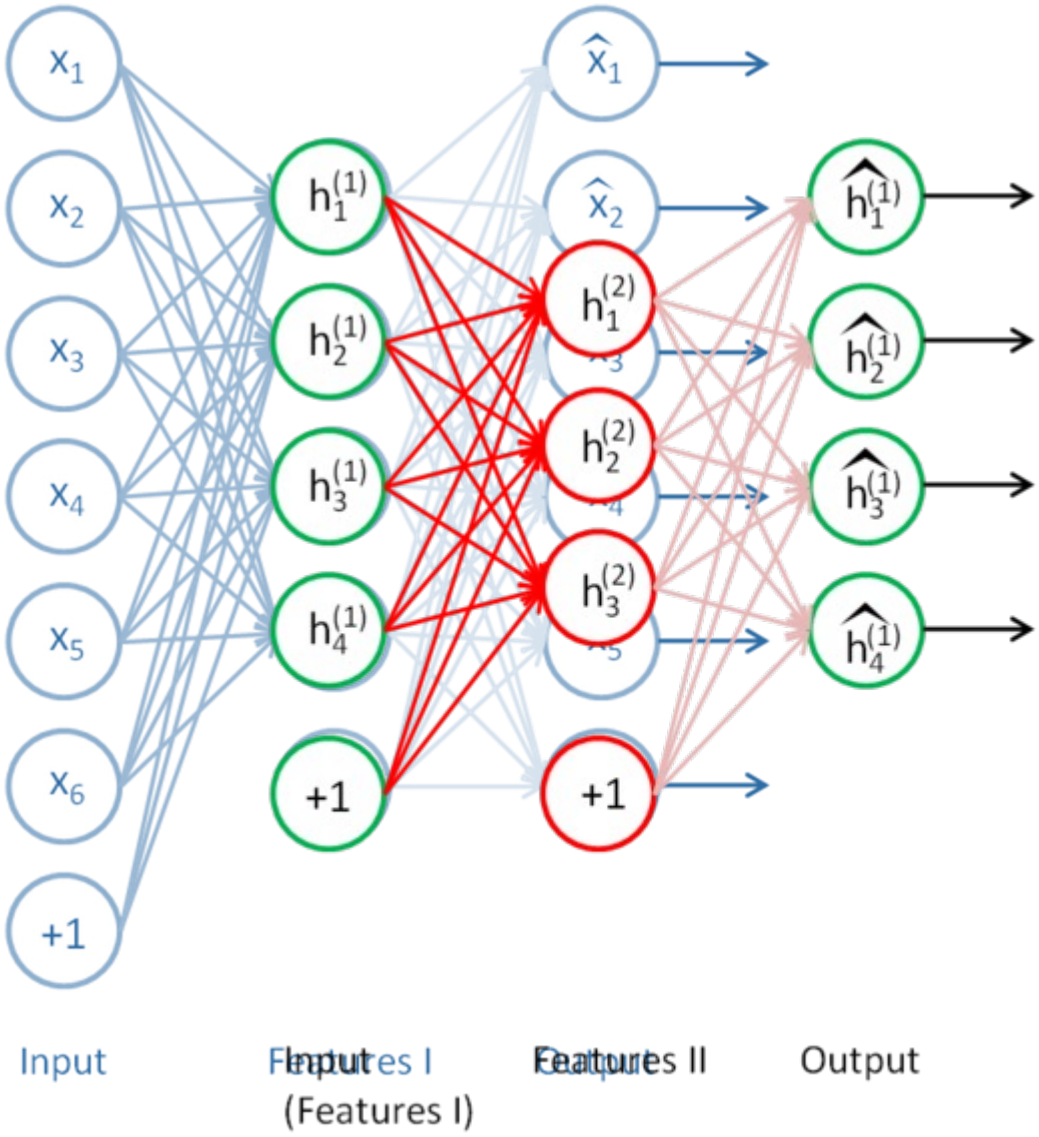
(Pascal Vincent)

Auto-encoder Approaches



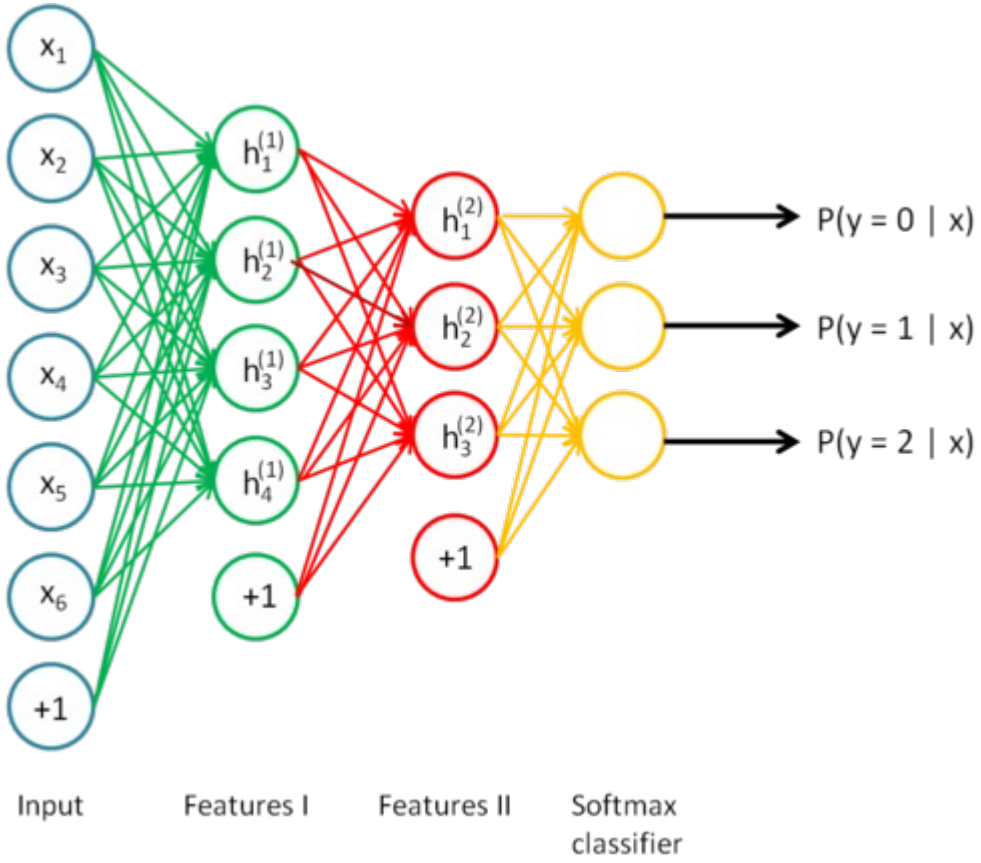
http://ufldl.stanford.edu/wiki/index.php/Stacked_Autoencoders

Auto-encoder Approaches: Second-layer



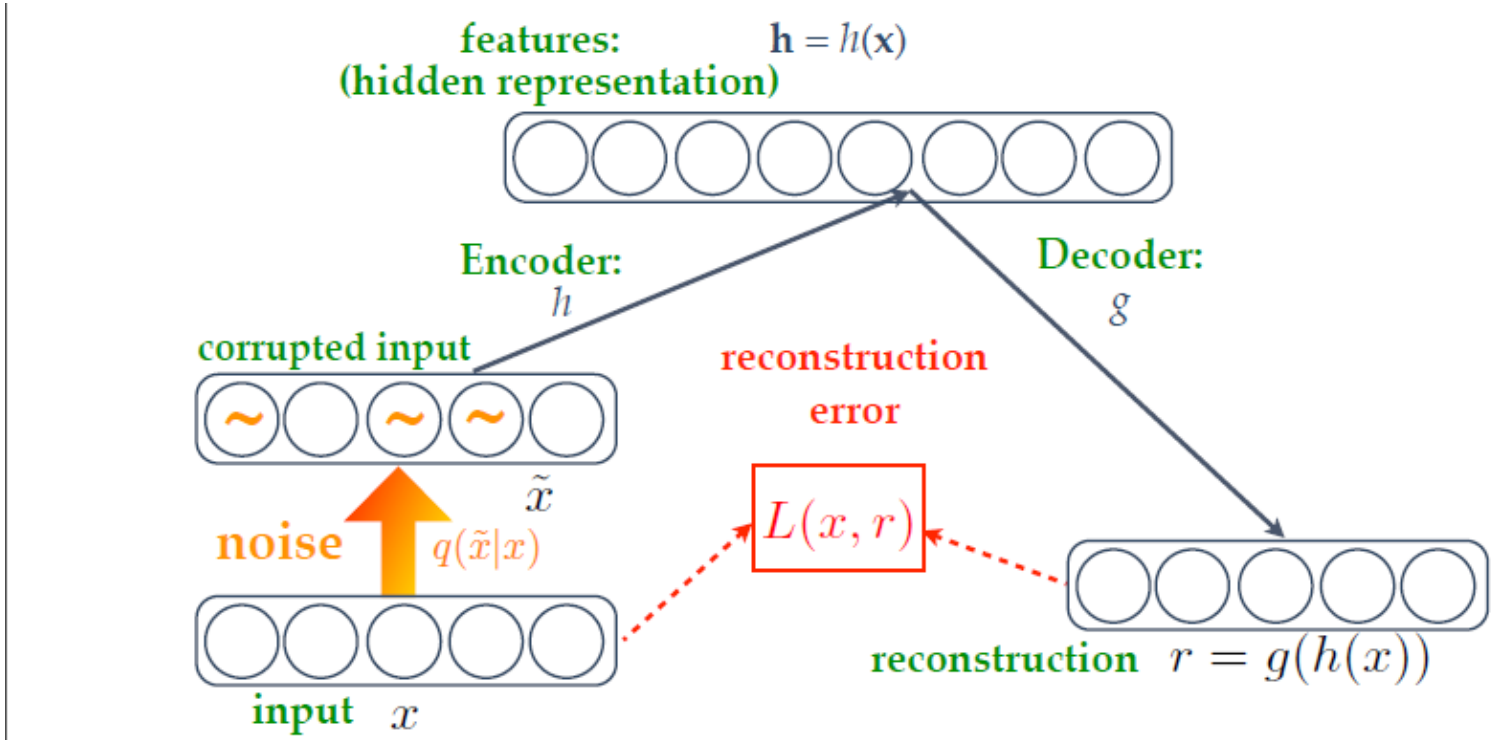
http://ufldl.stanford.edu/wiki/index.php/Stacked_Autoencoders

Autoencoder Approaches: Adding task layer



http://ufldl.stanford.edu/wiki/index.php/Stacked_Autoencoders

Autoencoder Approaches: Denoising AE



- learns robust & useful features
- easier to train than RBM features
- yield similar or better classification performance (as deep net pre-training)

Minimize:

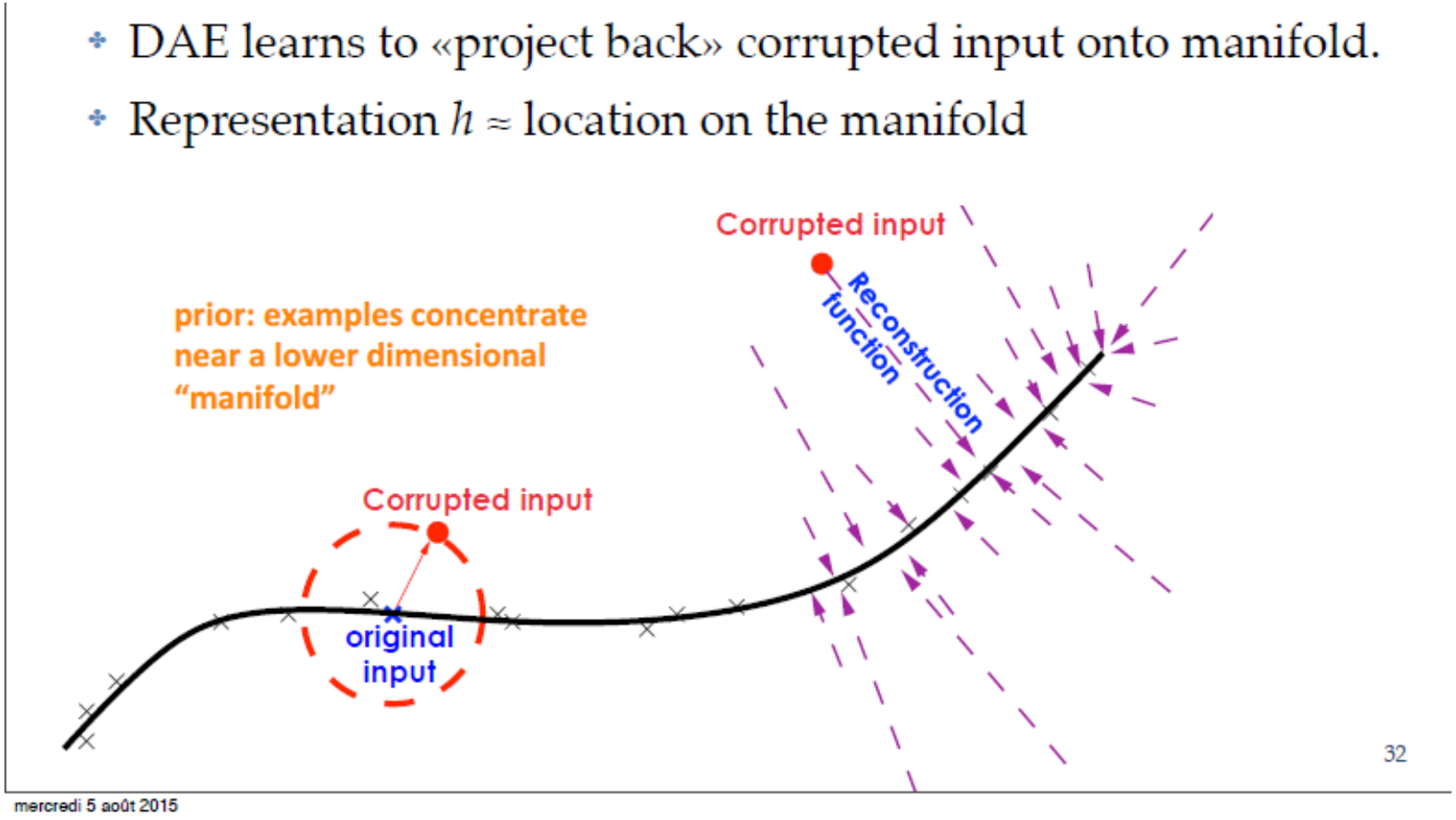
$$\mathcal{J}_{\mathrm{DAE}}(\theta) = \sum_{x \in D} \mathbb{E}_{q(\tilde{x}|x)} \left[L(x, g(h(\tilde{x}))) \right]$$

mercredi 5 août 2015

(Pascal Vincent)

Autoencoder Approaches: Denoising AE

- DAE learns to «project back» corrupted input onto manifold.
- Representation $h \approx$ location on the manifold



(Pascal Vincent)

Context-Prediction Approaches

- Word embeddings are the basis of deep learning for NLP

- Word embeddings (`word2vec`, `GloVe`) are often *pre-trained* on text corpus from co-occurrence statistics

Slide: Jacob Devlin
https://nlp.stanford.edu/seminar/details/jdevlin.pdf

Constrastive Approaches

Siamese Networks

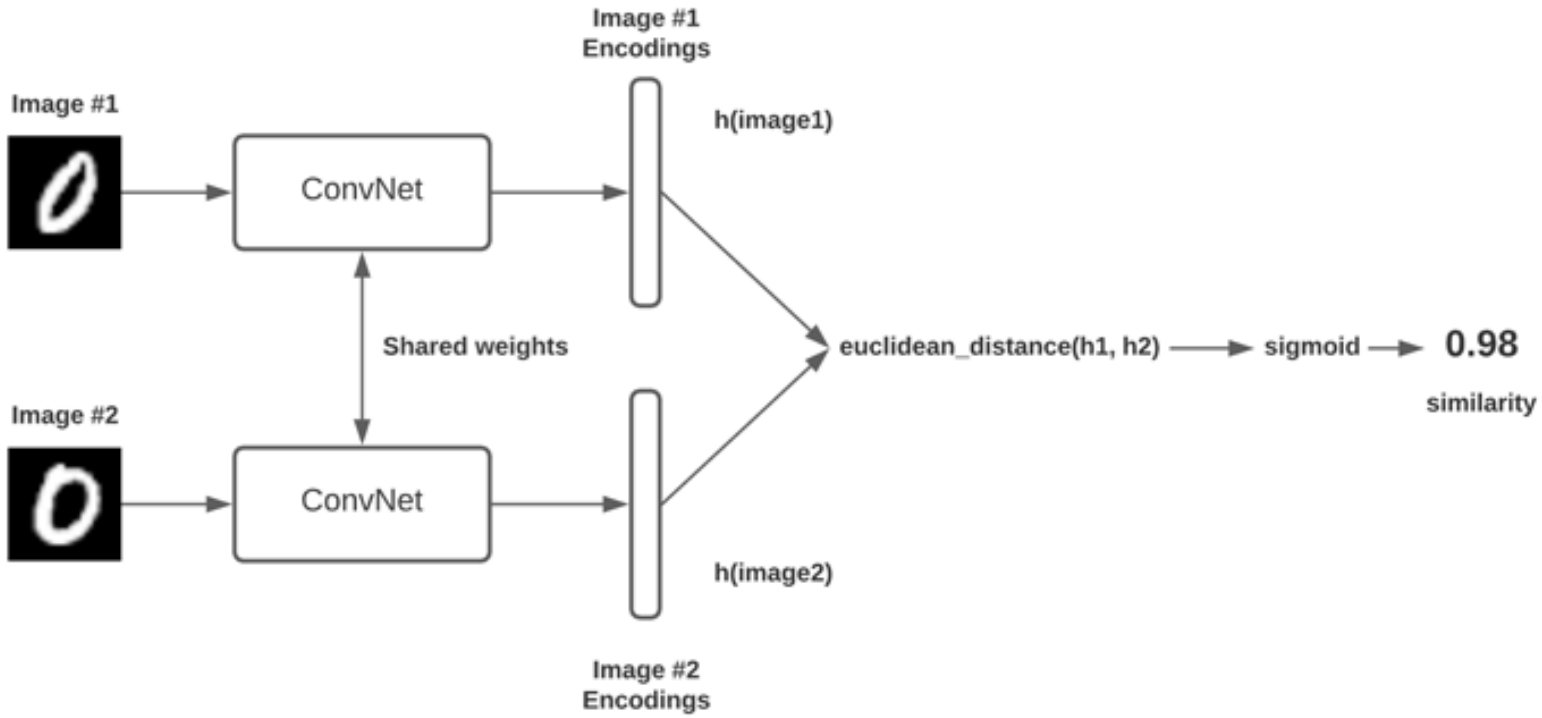


Fig: https://www.pyimagesearch.com/2020/11/30/siamese-networks-with-keras-tensorflow-and-deep-learning/

Contrastive Loss (Chopra et al., 2005)

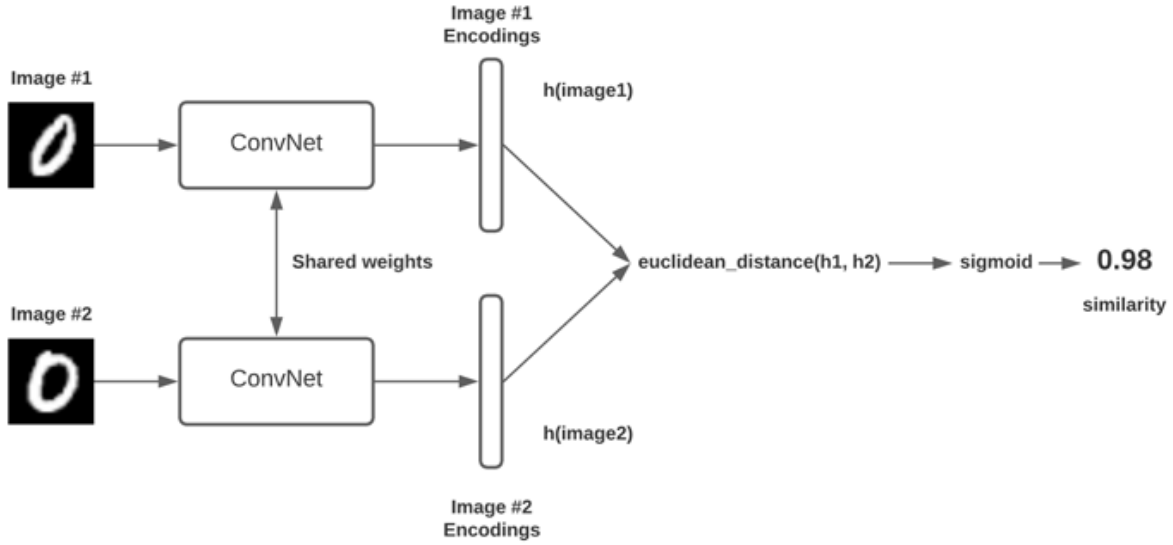


Fig: https://www.pyimagesearch.com/2020/11/30/siamese-networks-with-keras-tensorflow-and-deep-learning/

$y_i = y_j$ for "similar" pairs:

$$\mathcal{L}_{\text{cont}}(\mathbf{x}_i, \mathbf{x}_j, \theta) = \mathbb{1}[y_i = y_j] \|f_\theta(\mathbf{x}_i) - f_\theta(\mathbf{x}_j)\|_2^2 + \mathbb{1}[y_i \neq y_j] \max(0, \epsilon - \|f_\theta(\mathbf{x}_i) - f_\theta(\mathbf{x}_j)\|_2)^2$$

similar pairs

dissimilar pairs

From: https://lilianweng.github.io/posts/2021-05-31-contrastive/

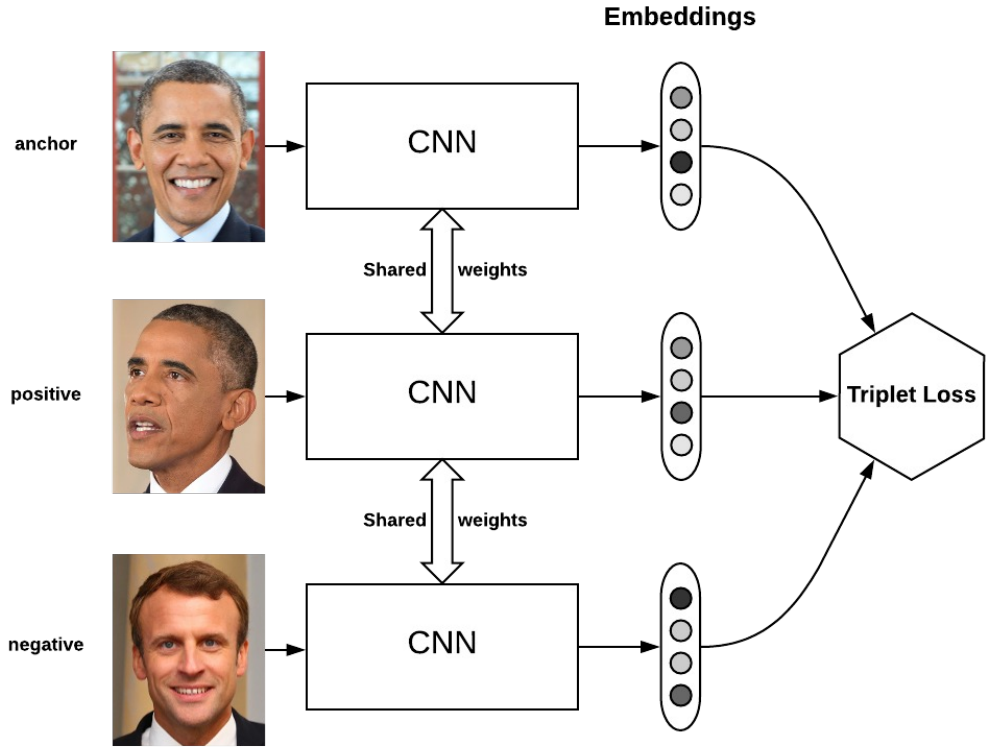
Triplet Loss (Schroff et al., 2015)

Fig. 1. Illustration of triplet loss given one positive and one negative per anchor. (Image source: Schroff et al. 2015)

$$\mathcal{L}_{\text{triplet}}(\mathbf{x}, \mathbf{x}^+, \mathbf{x}^-) = \sum_{\mathbf{x} \in \mathcal{X}} \max\left(0, \|f(\mathbf{x}) - f(\mathbf{x}^+)\|_2^2 - \|f(\mathbf{x}) - f(\mathbf{x}^-)\|_2^2 + \epsilon\right)$$

anchor-positive distance

anchor-negative distance



https://omoindrot.github.io/triplet-loss

From: https://lilianweng.github.io/posts/2021-05-31-contrastive/

Lifted Structure Loss (Song et al., 2015)

Let $D_{ij} = \|f(\mathbf{x}_i) - f(\mathbf{x}_j)\|_2$, a structured loss function is defined as

$$\mathcal{L}_{\text{struct}} = \frac{1}{2|\mathcal{P}|} \sum_{(i,j)\in\mathcal{P}} \max(0, \mathcal{L}_{\text{struct}}^{(ij)})^2$$

$$\text{where } \mathcal{L}_{\text{struct}}^{(ij)} = D_{ij} + \max\left(\max_{(i,k)\in\mathcal{N}} \epsilon - D_{ik}, \max_{(j,l)\in\mathcal{N}} \epsilon - D_{jl} \right)$$

positive-positive distance

positive-negative distance

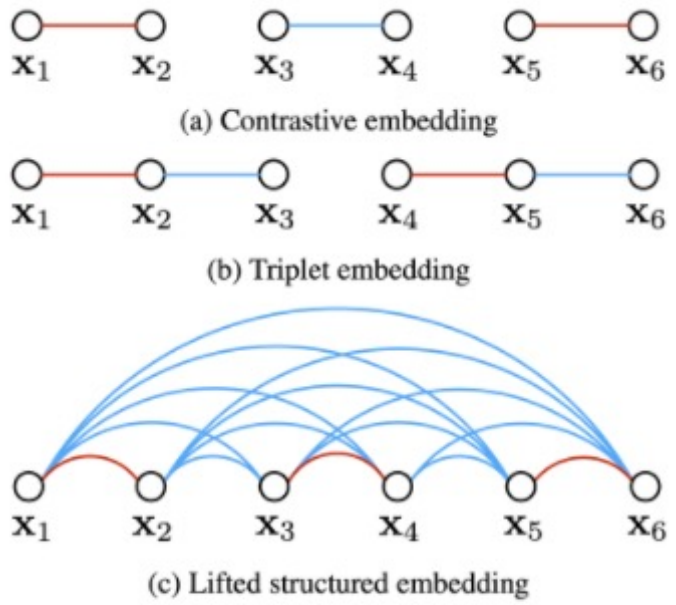


Fig. 2. Illustration compares contrastive loss, triplet loss and lifted structured loss. Red and blue edges connect similar and dissimilar sample pairs respectively. (Image source: Song et al. 2015)

From: https://lilianweng.github.io/posts/2021-05-31-contrastive/

N-pair Loss (Sohn 2016)

$$\mathcal{L}_{\text{N-pair}}(\mathbf{x}, \mathbf{x}^+, \{\mathbf{x}_i^-\}_{i=1}^{N-1}) = \log\left(1 + \sum_{i=1}^{N-1} \exp(f(\mathbf{x})^\top f(\mathbf{x}_i^-) - f(\mathbf{x})^\top f(\mathbf{x}^+))\right)$$

$$= -\log \frac{\exp(f(\mathbf{x})^\top f(\mathbf{x}^+))}{\exp(f(\mathbf{x})^\top f(\mathbf{x}^+)) + \sum_{i=1}^{N-1} \exp(f(\mathbf{x})^\top f(\mathbf{x}_i^-))}$$

positive-positive distance

positive-positive distance

positive-negative distance

From: https://lilianweng.github.io/posts/2021-05-31-contrastive/

InfoNCE Loss (van den Oord et al., 2018)

The InfoNCE loss optimizes the negative log probability of classifying the positive sample correctly:

$$\mathcal{L}_{\text{InfoNCE}} = -\mathbb{E}\left[\log \frac{f(\mathbf{x}, \mathbf{c})}{\sum_{\mathbf{x}' \in X} f(\mathbf{x}', \mathbf{c})}\right]$$

$$f(\mathbf{x}, \mathbf{c}) \propto \frac{p(\mathbf{x}|\mathbf{c})}{p(\mathbf{x})}$$

From: https://lilianweng.github.io/posts/2021-05-31-contrastive/

Momentum Contrast (MoCo)

- Contrastive learning as dictionary lookup:

for q). With similarity measured by dot product, a form of a contrastive loss function, called InfoNCE [46], is considered in this paper:

$$\mathcal{L}_q = -\log \frac{\exp(q \cdot k_+/\tau)}{\sum_{i=0}^{K} \exp(q \cdot k_i/\tau)} \quad (1)$$

where τ is a temperature hyper-parameter per [61]. The sum is over one positive and K negative samples. Intuitively, this loss is the log loss of a $(K+1)$-way softmax-based classifier that tries to classify q as k_+. Contrastive loss functions

Momentum Contrast for Unsupervised Visual Representation Learning

Kaiming He Haoqi Fan Yuxin Wu Saining Xie Ross Girshick 2019

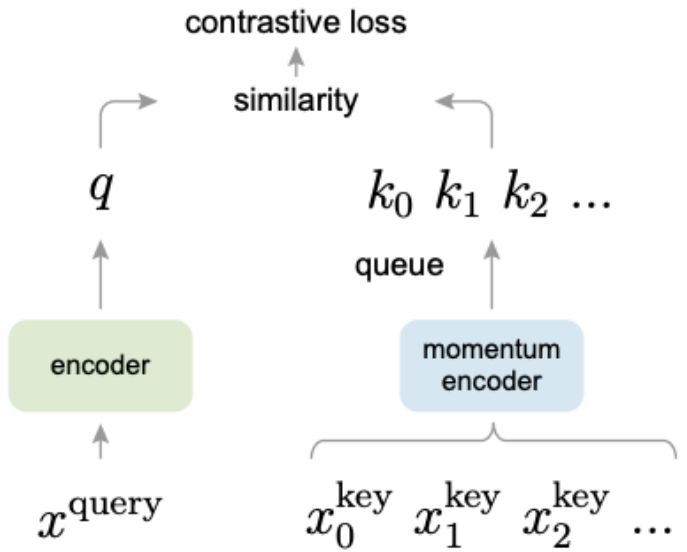


Figure 1. Momentum Contrast (MoCo) trains a visual representation encoder by matching an encoded query q to a dictionary of encoded keys using a contrastive loss. The dictionary keys $\{k_0, k_1, k_2, ...\}$ are defined on-the-fly by a set of data samples. The dictionary is built as a queue, with the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

Momentum Contrast for Unsupervised Visual Representation Learning

Kaiming He Haoqi Fan Yuxin Wu Saining Xie Ross Girshick 2019

Momentum Contrast

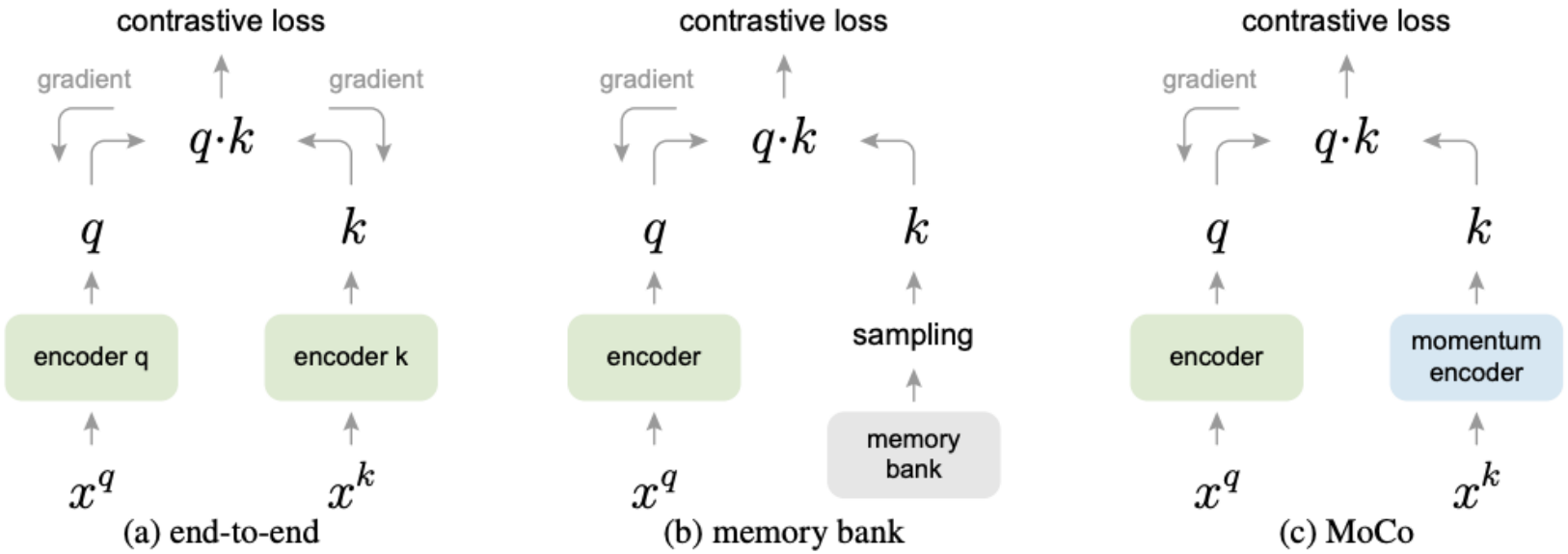


Figure 2. **Conceptual comparison of three contrastive loss mechanisms** (empirical comparisons are in Figure 3 and Table 3). Here we illustrate one pair of query and key. The three mechanisms differ in how the keys are maintained and how the key encoder is updated. **(a)**: The encoders for computing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

Momentum Contrast

- Dictionary
  - A queue of data samples
  - Encoded keys from immediately preceding mini-batches
  - Decouples dictionary size from batchsize
  - Samples are progressively replaced: Current batch is added and the oldest is removed.

Momentum Contrast for Unsupervised Visual Representation Learning

Kaiming He Haoqi Fan Yuxin Wu Saining Xie Ross Girshick 2019

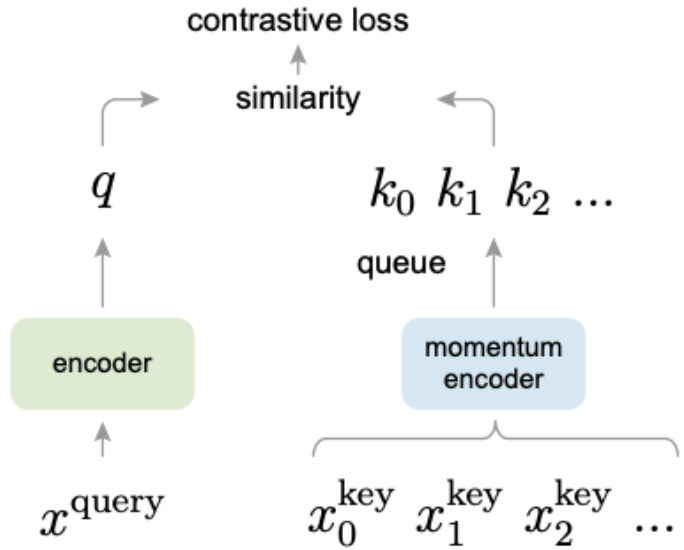


Figure 1. Momentum Contrast (MoCo) trains a visual representation encoder by matching an encoded query q to a dictionary of encoded keys using a contrastive loss. The dictionary keys $\{k_0, k_1, k_2, ...\}$ are defined on-the-fly by a set of data samples. The dictionary is built as a queue, with the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

Momentum Contrast

Momentum Contrast for Unsupervised Visual Representation Learning

Kaiming He Haoqi Fan Yuxin Wu Saining Xie Ross Girshick 2019

- Momentum update
  - Queue size can be a limiting factor especially if we backpropagate to the samples in the queue as well
  - Naïve solution: Copy image encoder to the queue encoder => Does not work well.
  - Effective solution: Update the queue encoder with the image encoder with momentum update

Formally, denoting the parameters of f_k as θ_k and those of f_q as θ_q, we update θ_k by:

$$\theta_k \leftarrow m\theta_k + (1-m)\theta_q. \quad (2)$$

Here $m \in [0, 1)$ is a momentum coefficient. Only the parameters θ_q are updated by back-propagation. The momen-

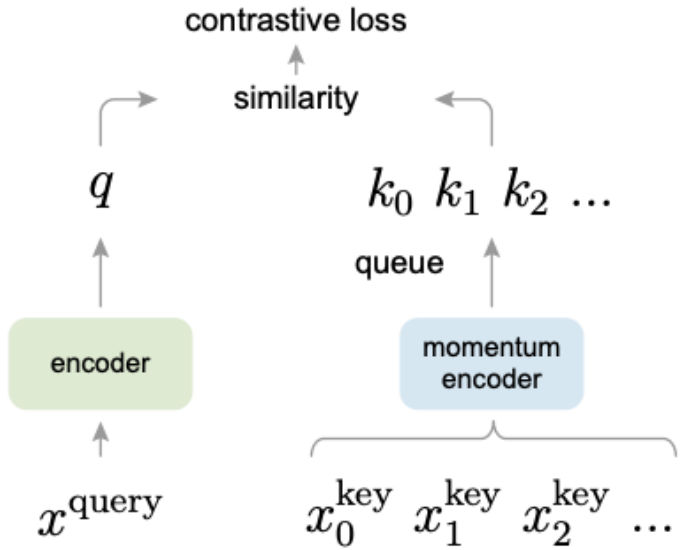


Figure 1. Momentum Contrast (MoCo) trains a visual representation encoder by matching an encoded query q to a dictionary of encoded keys using a contrastive loss. The dictionary keys $\{k_0, k_1, k_2, ...\}$ are defined on-the-fly by a set of data samples. The dictionary is built as a queue, with the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

Momentum Contrast

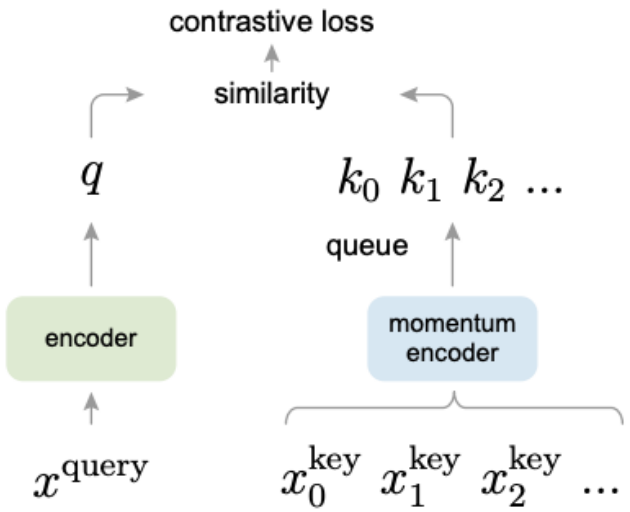


Figure 1. Momentum Contrast (MoCo) trains a visual representation encoder by matching an encoded query q to a dictionary of encoded keys using a contrastive loss. The dictionary keys $\{k_0, k_1, k_2, ...\}$ are defined on-the-fly by a set of data samples. The dictionary is built as a queue, with the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

Momentum Contrast for Unsupervised Visual Representation Learning

Kaiming He Haoqi Fan Yuxin Wu Saining Xie Ross Girshick 2019

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

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

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

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

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

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

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

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

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

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

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

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

Momentum Contrast

Momentum Contrast for Unsupervised Visual Representation Learning

Kaiming He Haoqi Fan Yuxin Wu Saining Xie Ross Girshick 2019

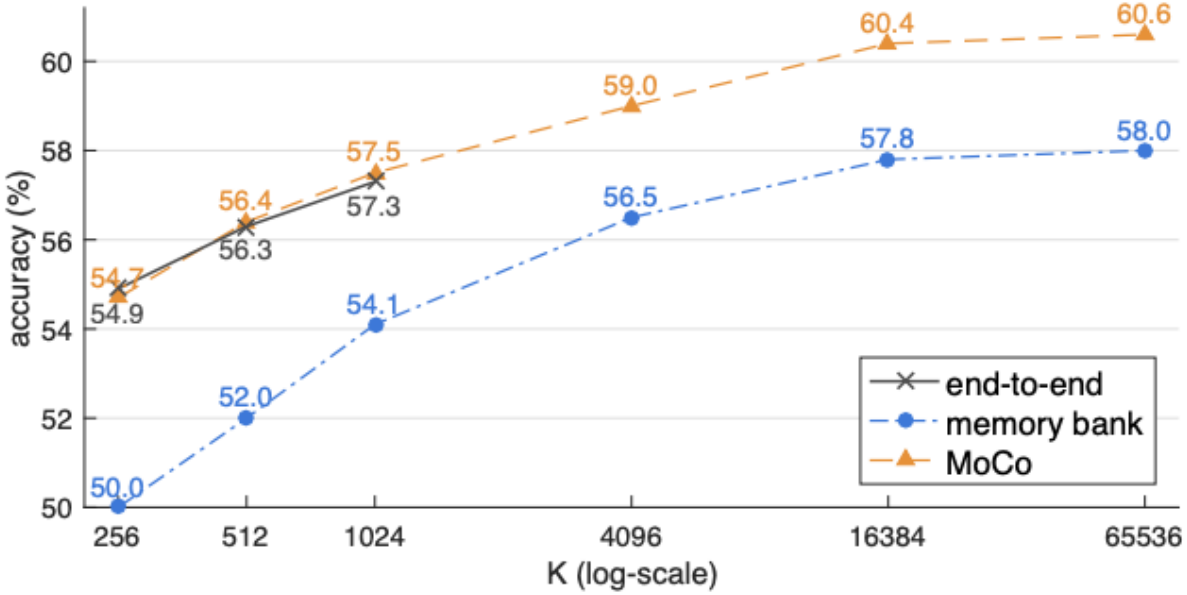


Figure 3. **Comparison of three contrastive loss mechanisms** under the ImageNet linear classification protocol. We adopt the same pretext task (Sec. 3.3) and only vary the contrastive loss mechanism (Figure 2). The number of negatives is K in memory bank and MoCo, and is $K-1$ in end-to-end (offset by one because the positive key is in the same mini-batch). The network is ResNet-50.

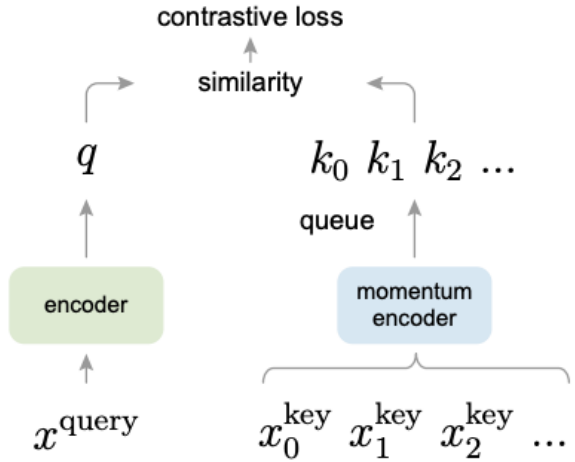


Figure 1. Momentum Contrast (MoCo) trains a visual representation encoder by matching an encoded query q to a dictionary of encoded keys using a contrastive loss. The dictionary keys $\{k_0, k_1, k_2, ...\}$ are defined on-the-fly by a set of data samples. The dictionary is built as a queue, with the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

Momentum Contrast

Momentum Contrast for Unsupervised Visual Representation Learning

Kaiming He Haoqi Fan Yuxin Wu Saining Xie Ross Girshick 2019

pre-train	R50-dilated-C5 AP_{50}	R50-dilated-C5 AP	R50-dilated-C5 AP_{75}	R50-C4 AP_{50}	R50-C4 AP	R50-C4 AP_{75}
end-to-end	79.2	52.0	56.6	80.4	54.6	60.3
memory bank	79.8	52.9	57.9	80.6	54.9	60.6
MoCo	**81.1**	**54.6**	**59.9**	**81.5**	**55.9**	**62.6**

Table 3. **Comparison of three contrastive loss mechanisms** on PASCAL VOC object detection, fine-tuned on `trainval07+12` and evaluated on `test2007` (averages over 5 trials). All models are implemented by us (Figure 3), pre-trained on IN-1M, and fine-tuned using the same settings as in Table 2.

pre-train	COCO keypoint detection AP^{kp}	AP^{kp}_{50}	AP^{kp}_{75}
random init.	65.9	86.5	71.7
super. IN-1M	65.8	86.9	71.9
MoCo IN-1M	66.8 (+1.0)	87.4 (+0.5)	72.5 (+0.6)
MoCo IG-1B	66.9 (+1.1)	87.8 (+0.9)	73.0 (+1.1)

pre-train	COCO dense pose estimation AP^{dp}	AP^{dp}_{50}	AP^{dp}_{75}
random init.	39.4	78.5	35.1
super. IN-1M	48.3	85.6	50.6
MoCo IN-1M	50.1 (+1.8)	86.8 (+1.2)	53.9 (+3.3)
MoCo IG-1B	50.6 (+2.3)	87.0 (+1.4)	54.3 (+3.7)

pre-train	LVIS v0.5 instance segmentation AP^{mk}	AP^{mk}_{50}	AP^{mk}_{75}
random init.	22.5	34.8	23.8
super. IN-1M†	24.4	37.8	25.8
MoCo IN-1M	24.1 (−0.3)	37.4 (−0.4)	25.5 (−0.3)
MoCo IG-1B	24.9 (+0.5)	38.2 (+0.4)	26.4 (+0.6)

pre-train	Cityscapes instance seg. AP^{mk}	Cityscapes instance seg. AP^{mk}_{50}	Semantic seg. (mIoU) Cityscapes	Semantic seg. (mIoU) VOC
random init.	25.4	51.1	65.3	39.5
super. IN-1M	32.9	59.6	74.6	74.4
MoCo IN-1M	32.3 (−0.6)	59.3 (−0.3)	75.3 (+0.7)	72.5 (−1.9)
MoCo IG-1B	32.9 (0.0)	60.3 (+0.7)	75.5 (+0.9)	73.6 (−0.8)

Table 6. **MoCo *vs.* ImageNet supervised pre-training, fine-tuned on various tasks**. For each task, the same architecture and schedule are used for all entries (see appendix). In the brackets are the gaps to the ImageNet supervised pre-training counterpart. In green are the gaps of at least +0.5 point.

Simple Contrastive Learning

A Simple Framework for Contrastive Learning of Visual Representations

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

2020

as negative examples. Let $\text{sim}(\boldsymbol{u}, \boldsymbol{v}) = \boldsymbol{u}^\top \boldsymbol{v}/\|\boldsymbol{u}\|\|\boldsymbol{v}\|$ denote the dot product between ℓ_2 normalized $\boldsymbol{u}$ and $\boldsymbol{v}$ (i.e. cosine similarity). Then the loss function for a positive pair of examples (i, j) is defined as

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

where $\mathbb{1}_{[k \neq i]} \in \{0, 1\}$ is an indicator function evaluating to 1 iff $k \neq i$ and τ denotes a temperature parameter. The fi-

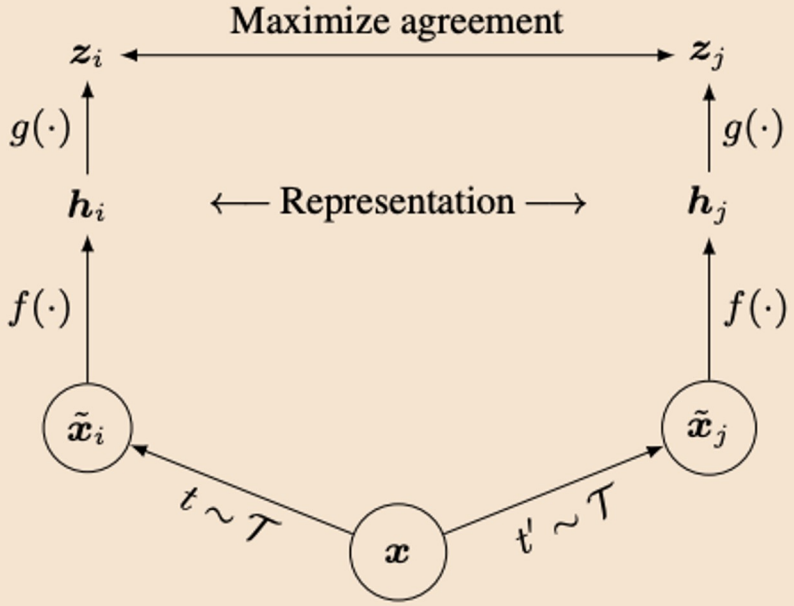


Figure 2. A simple framework for contrastive learning of visual representations. Two separate data augmentation operators are sampled from the same family of augmentations ($t \sim \mathcal{T}$ and $t' \sim \mathcal{T}$) and applied to each data example to obtain two correlated views. A base encoder network $f(\cdot)$ and a projection head $g(\cdot)$ are trained to maximize agreement using a contrastive loss. After training is completed, we throw away the projection head $g(\cdot)$ and use encoder $f(\cdot)$ and representation $\boldsymbol{h}$ for downstream tasks.

Simple Contrastive Learning

Algorithm 1 SimCLR's main learning algorithm.

input: batch size N, constant τ, structure of $f, g, \mathcal{T}$.
for sampled minibatch $\{\boldsymbol{x}_k\}_{k=1}^{N}$ **do**
  for all $k \in \{1, \dots, N\}$ **do**
    draw two augmentation functions $t \sim \mathcal{T}, t' \sim \mathcal{T}$
    # the first augmentation
    $\tilde{\boldsymbol{x}}_{2k-1} = t(\boldsymbol{x}_k)$
    $\boldsymbol{h}_{2k-1} = f(\tilde{\boldsymbol{x}}_{2k-1})$ # representation
    $\boldsymbol{z}_{2k-1} = g(\boldsymbol{h}_{2k-1})$ # projection
    # the second augmentation
    $\tilde{\boldsymbol{x}}_{2k} = t'(\boldsymbol{x}_k)$
    $\boldsymbol{h}_{2k} = f(\tilde{\boldsymbol{x}}_{2k})$ # representation
    $\boldsymbol{z}_{2k} = g(\boldsymbol{h}_{2k})$ # projection
  end for
  for all $i \in \{1, \dots, 2N\}$ and $j \in \{1, \dots, 2N\}$ **do**
    $s_{i,j} = \boldsymbol{z}_i^\top \boldsymbol{z}_j / (\|\boldsymbol{z}_i\| \|\boldsymbol{z}_j\|)$ # pairwise similarity
  end for
  define $\ell(i, j)$ **as** $\ell(i, j) = -\log \frac{\exp(s_{i,j}/\tau)}{\sum_{k=1}^{2N} \mathbb{1}_{[k \neq i]} \exp(s_{i,k}/\tau)}$
  $\mathcal{L} = \frac{1}{2N} \sum_{k=1}^{N} [\ell(2k-1, 2k) + \ell(2k, 2k-1)]$
  update networks f and g to minimize $\mathcal{L}$
end for
return encoder network $f(\cdot)$, and throw away $g(\cdot)$

A Simple Framework for Contrastive Learning of Visual Representations

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

2020

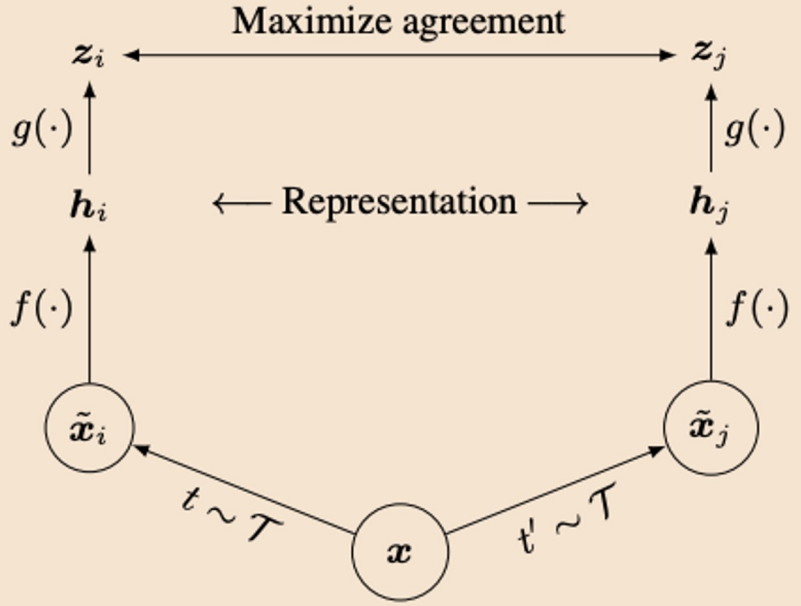


Figure 2. A simple framework for contrastive learning of visual representations. Two separate data augmentation operators are sampled from the same family of augmentations ($t \sim \mathcal{T}$ and $t' \sim \mathcal{T}$) and applied to each data example to obtain two correlated views. A base encoder network $f(\cdot)$ and a projection head $g(\cdot)$ are trained to maximize agreement using a contrastive loss. After training is completed, we throw away the projection head $g(\cdot)$ and use encoder $f(\cdot)$ and representation $\boldsymbol{h}$ for downstream tasks.

Simple Contrastive Learning

A Simple Framework for Contrastive Learning of Visual Representations

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

2020

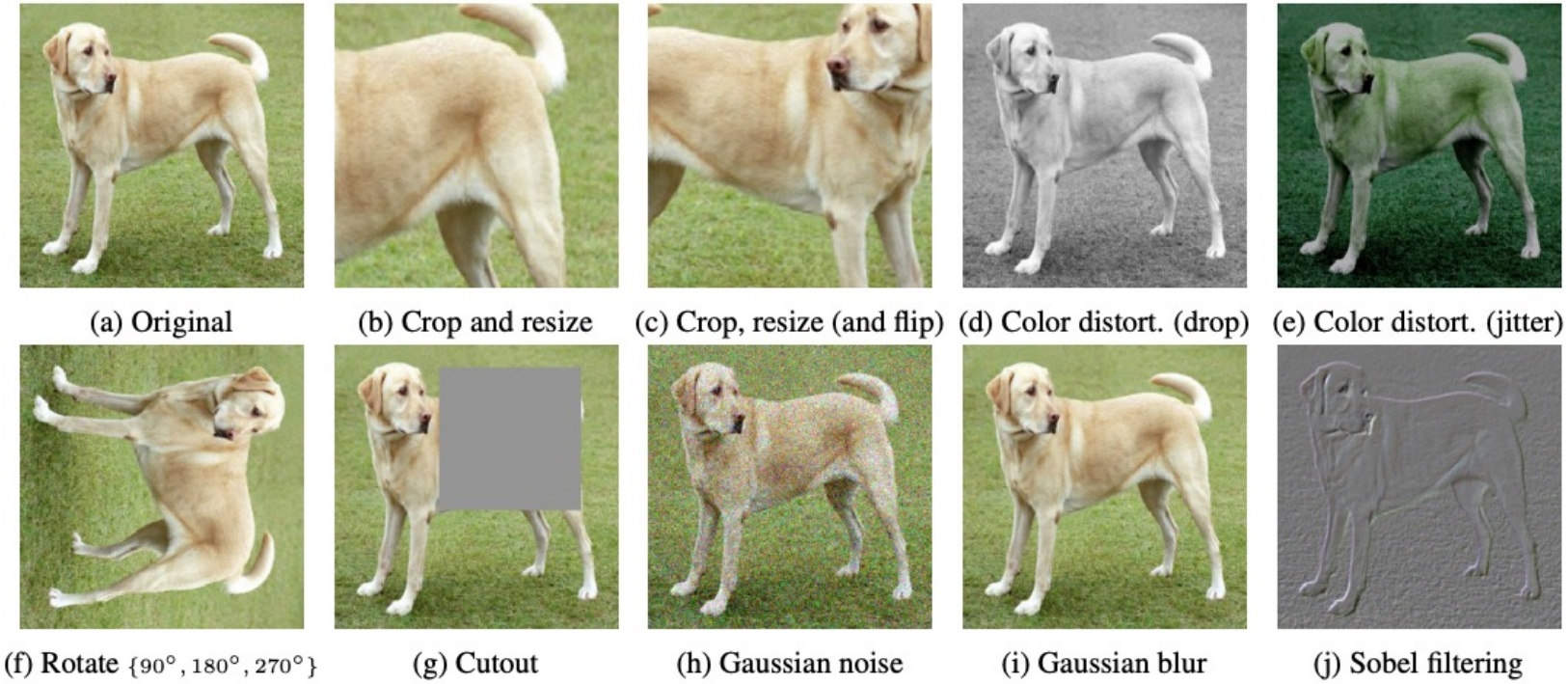


Figure 4. Illustrations of the studied data augmentation operators. Each augmentation can transform data stochastically with some internal parameters (e.g. rotation degree, noise level). Note that we *only* test these operators in ablation, the *augmentation policy used to train our models* only includes *random crop (with flip and resize)*, *color distortion*, and *Gaussian blur*. (Original image cc-by: Von.grzanka)

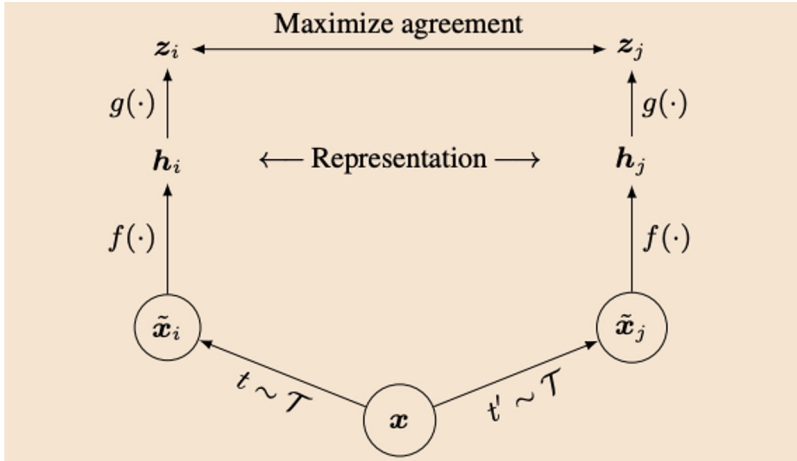


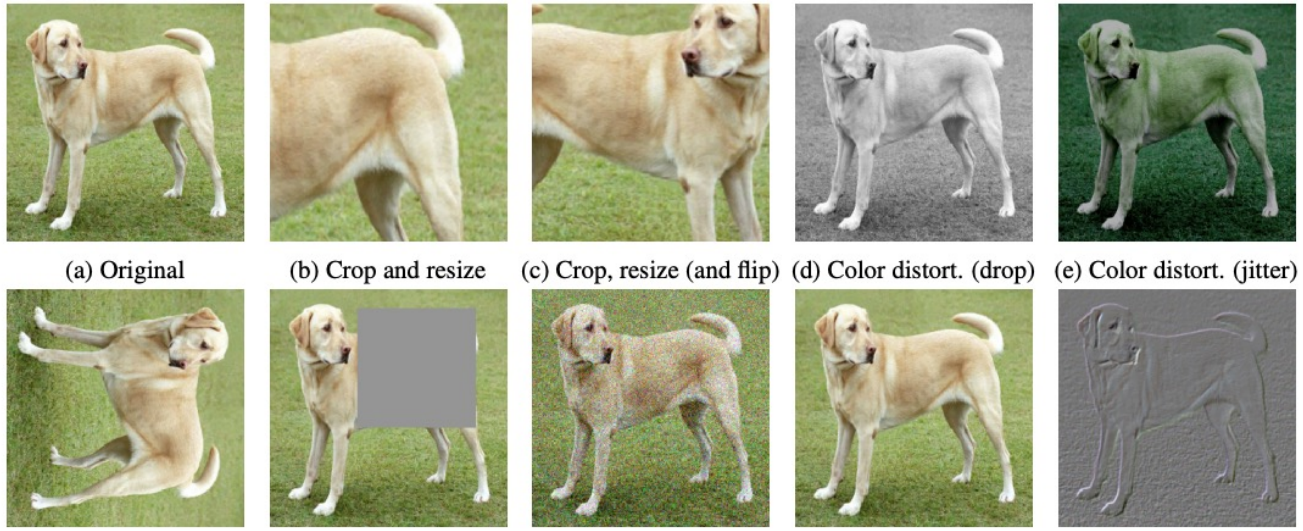
Figure 2. A simple framework for contrastive learning of visual representations. Two separate data augmentation operators are sampled from the same family of augmentations ($t \sim \mathcal{T}$ and $t' \sim \mathcal{T}$) and applied to each data example to obtain two correlated views. A base encoder network $f(\cdot)$ and a projection head $g(\cdot)$ are trained to maximize agreement using a contrastive loss. After training is completed, we throw away the projection head $g(\cdot)$ and use encoder $f(\cdot)$ and representation $\boldsymbol{h}$ for downstream tasks.

Simple Contrastive Learning

A Simple Framework for Contrastive Learning of Visual Representations

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

2020



(a) Original (b) Crop and resize (c) Crop, resize (and flip) (d) Color distort. (drop) (e) Color distort. (jitter)

(f) Rotate $\{90°, 180°, 270°\}$ (g) Cutout (h) Gaussian noise (i) Gaussian blur (j) Sobel filtering

Figure 4. Illustrations of the studied data augmentation operators. Each augmentation can transform data stochastically with some internal parameters (e.g. rotation degree, noise level). Note that we *only* test these operators in ablation, the *augmentation policy used to train our models* only includes *random crop (with flip and resize)*, *color distortion*, and *Gaussian blur*. (Original image cc-by: Von.grzanka)

Figure 5. Linear evaluation (ImageNet top-1 accuracy) under individual or composition of data augmentations, applied only to one branch. For all columns but the last, diagonal entries correspond to single transformation, and off-diagonals correspond to composition of two transformations (applied sequentially). The last column reflects the average over the row.

Simple Contrastive Learning

A Simple Framework for Contrastive Learning of Visual Representations

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

2020

Methods	Color distortion strength 1/8	1/4	1/2	1	1 (+Blur)	AutoAug
SimCLR	59.6	61.0	62.6	63.2	64.5	61.1
Supervised	77.0	76.7	76.5	75.7	75.4	77.1

Table 1. Top-1 accuracy of unsupervised ResNet-50 using linear evaluation and supervised ResNet-50[5], under varied color distortion strength (see Appendix A) and other data transformations. Strength 1 (+Blur) is our default data augmentation policy.

	Food	CIFAR10	CIFAR100	Birdsnap	SUN397	Cars	Aircraft	VOC2007	DTD	Pets	Caltech-101	Flowers
Linear evaluation:												
SimCLR (ours)	**76.9**	**95.3**	80.2	48.4	**65.9**	60.0	61.2	**84.2**	**78.9**	89.2	**93.9**	**95.0**
Supervised	75.2	**95.7**	**81.2**	**56.4**	64.9	**68.8**	**63.8**	83.8	**78.7**	**92.3**	**94.1**	94.2
Fine-tuned:												
SimCLR (ours)	**89.4**	**98.6**	**89.0**	**78.2**	**68.1**	**92.1**	**87.0**	**86.6**	**77.8**	92.1	**94.1**	97.6
Supervised	88.7	98.3	**88.7**	**77.8**	67.0	91.4	**88.0**	86.5	**78.8**	**93.2**	**94.2**	**98.0**
Random init	88.3	96.0	81.9	**77.0**	53.7	91.3	84.8	69.4	64.1	82.7	72.5	92.5

Table 8. Comparison of transfer learning performance of our self-supervised approach with supervised baselines across 12 natural image classification datasets, for ResNet-50 $(4\times)$ models pretrained on ImageNet. Results not significantly worse than the best ($p > 0.05$, permutation test) are shown in bold. See Appendix B.8 for experimental details and results with standard ResNet-50.

Simple Contrastive Learning

A Simple Framework for Contrastive Learning of Visual Representations

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

2020

- Requires large batchsize

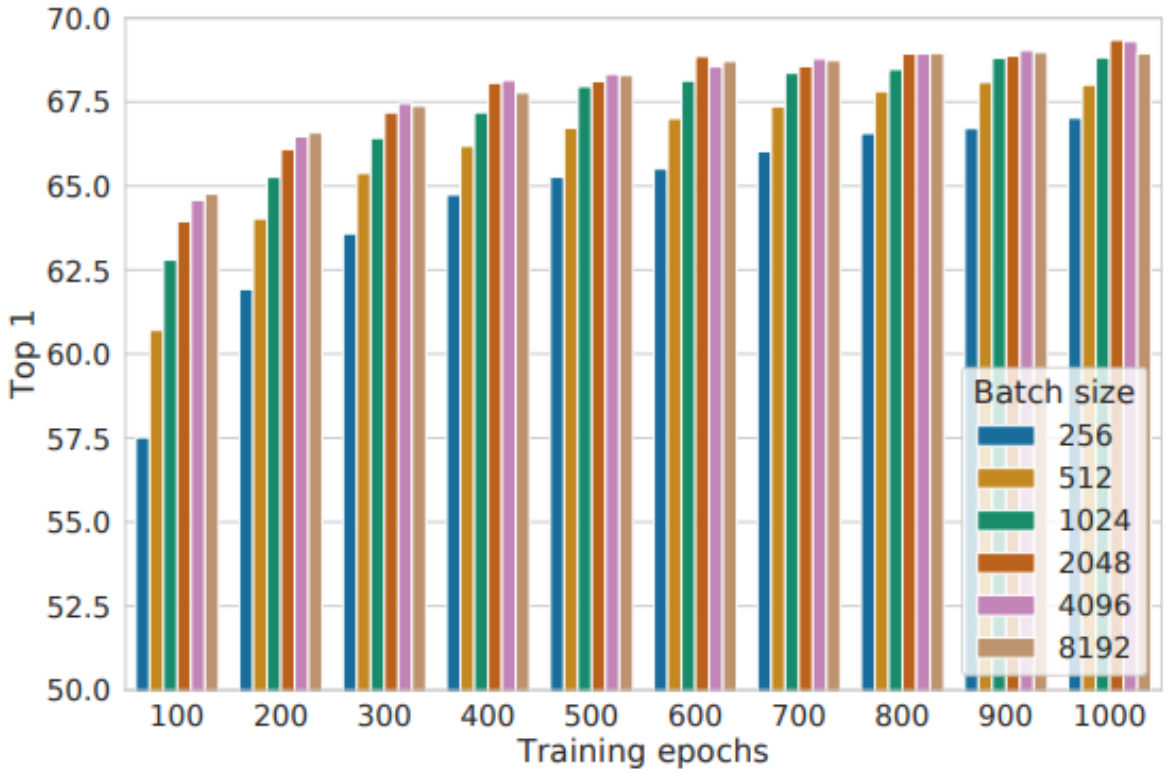


Figure 9. Linear evaluation models (ResNet-50) trained with different batch size and epochs. Each bar is a single run from scratch.[10]

MoCo v2

Improved Baselines with Momentum Contrastive Learning

Xinlei Chen Haoqi Fan Ross Girshick Kaiming He 2020

Abstract

Contrastive unsupervised learning has recently shown encouraging progress, e.g., in Momentum Contrast (MoCo) and SimCLR. In this note, we verify the effectiveness of two of SimCLR's design improvements by implementing them in the MoCo framework. With simple modifications to MoCo—namely, using an MLP projection head and more data augmentation—we establish stronger baselines that outperform SimCLR and do not require large training batches. We hope this will make state-of-the-art unsupervised learning research more accessible. Code will be made public.

case	unsup. pre-train MLP	aug+	cos	epochs	batch	ImageNet acc.
MoCo v1 [6]				200	256	60.6
SimCLR [2]	✓	✓	✓	200	256	61.9
SimCLR [2]	✓	✓	✓	200	8192	66.6
MoCo v2	✓	✓	✓	200	256	**67.5**
*results of **longer** unsupervised training follow:*						
SimCLR [2]	✓	✓	✓	1000	4096	69.3
MoCo v2	✓	✓	✓	800	256	**71.1**

Table 2. **MoCo *vs.* SimCLR**: ImageNet linear classifier accuracy (**ResNet-50, 1-crop 224×224**), trained on features from unsupervised pre-training. "aug+" in SimCLR includes blur and stronger color distortion. SimCLR ablations are from Fig. 9 in [2] (we thank the authors for providing the numerical results).

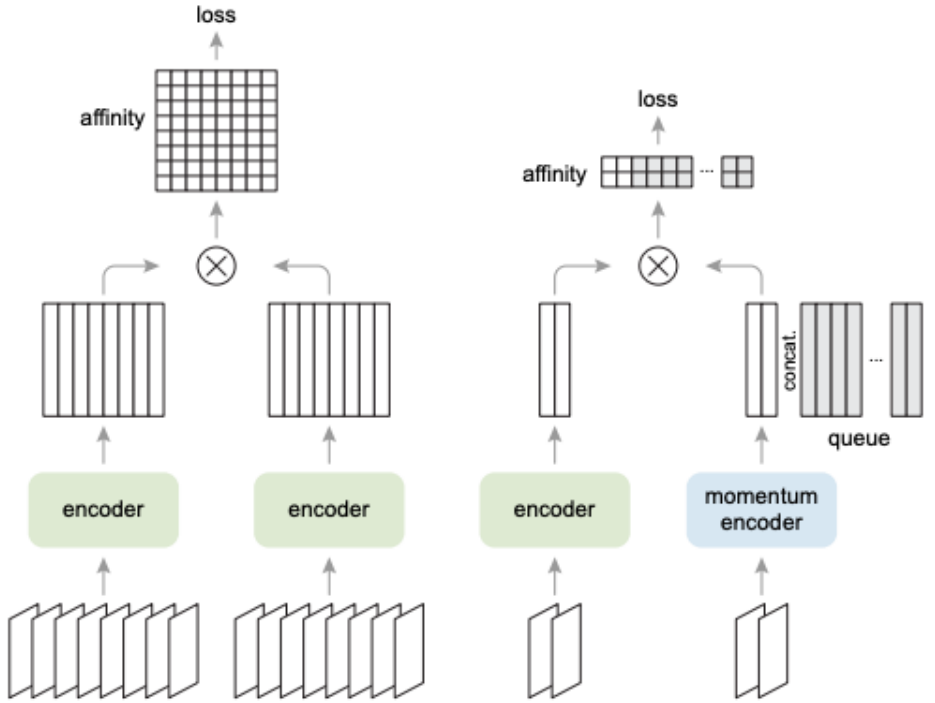


Figure 1. A **batching** perspective of two optimization mechanisms for contrastive learning. Images are encoded into a representation space, in which pairwise affinities are computed.

An Empirical Study of Training Self-Supervised Vision Transformers

Xinlei Chen* Saining Xie* Kaiming He 2021

MoCo v3 = MoCo v2 with ViT

framework	model	params	acc. (%)
linear probing:			
iGPT [9]	iGPT-L	1362M	69.0
iGPT [9]	iGPT-XL	6801M	72.0
MoCo v3	ViT-B	86M	76.7
MoCo v3	ViT-L	304M	77.6
MoCo v3	ViT-H	632M	78.1
MoCo v3	ViT-BN-H	632M	79.1
MoCo v3	ViT-BN-L/7	304M	**81.0**
end-to-end fine-tuning:			
masked patch pred. [16]	ViT-B	86M	79.9†
MoCo v3	ViT-B	86M	83.2
MoCo v3	ViT-L	304M	**84.1**

Table 1. **State-of-the-art Self-supervised Transformers** in ImageNet classification, evaluated by linear probing (top panel) or end-to-end fine-tuning (bottom panel). Both iGPT [9] and masked patch prediction [16] belong to the masked auto-encoding paradigm. MoCo v3 is a contrastive learning method that compares two (224×224) crops. ViT-B, -L, -H are the Vision Transformers proposed in [16]. ViT-BN is modified with BatchNorm, and "/7" denotes a patch size of 7×7. †: pre-trained in JFT-300M.

VICReg

Bardes et al., VICReg: Variance-Invariance-Covariance Regularization for Self-Supervised Learning, 2021.

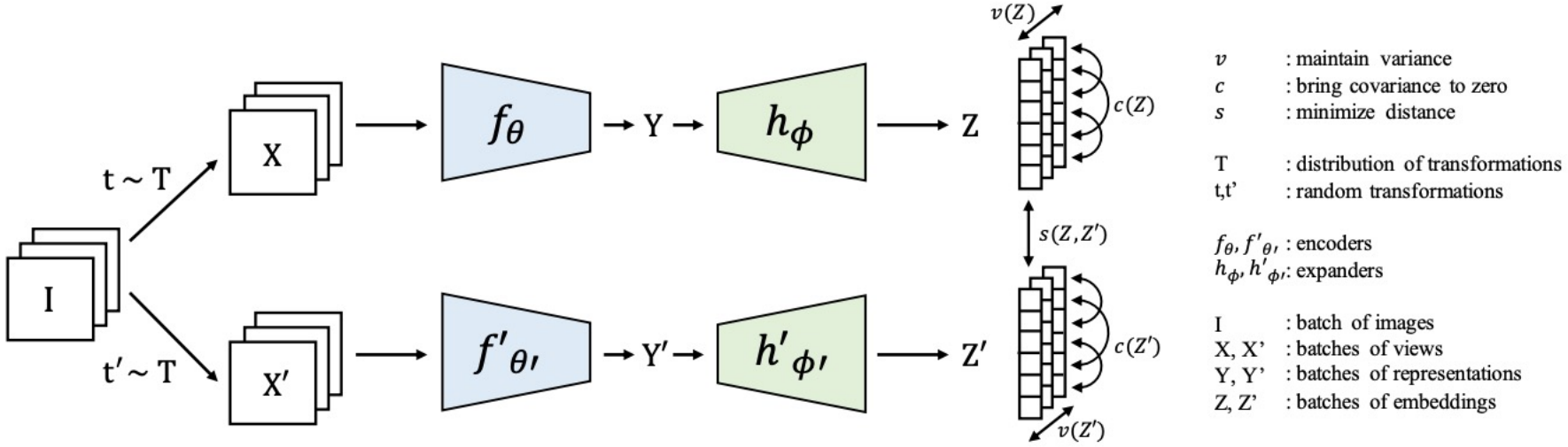
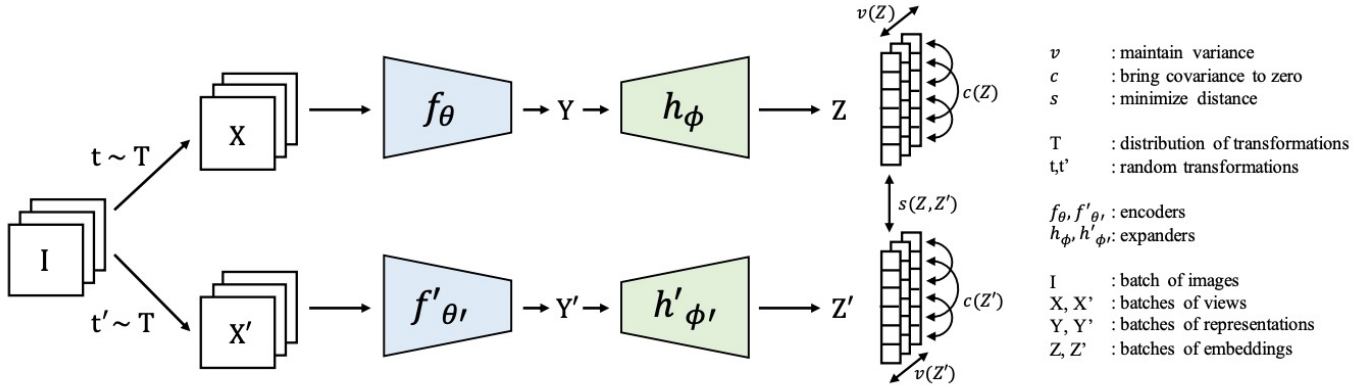


Figure 1: **VICReg: joint embedding architecture with variance, invariance and covariance regularization.** Given a batch of images I, two batches of different views X and X' are produced and are then encoded into representations Y and Y'. The representations are fed to an expander producing the embeddings Z and Z'. The distance between two embeddings from the same image is minimized, the variance of each embedding variable over a batch is maintained above a threshold, and the covariance between pairs of embedding variables over a batch are attracted to zero, decorrelating the variables from each other. Although the two branches do not require identical architectures nor share weights, in most of our experiments, they are Siamese with shared weights: the encoders are ResNet-50 backbones with output dimension 2048. The expanders have 3 fully-connected layers of size 8192.

VICReg



- **Invariance**: the mean square distance between the embedding vectors.
- **Variance**: a hinge loss to maintain the standard deviation (over a batch) of each variable of the embedding above a given threshold. This term forces the embedding vectors of samples within a batch to be different.
- **Covariance**: a term that attracts the covariances (over a batch) between every pair of (centered) embedding variables towards zero. This term decorrelates the variables of each embedding and prevents an *informational collapse* in which the variables would vary together or be highly correlated.

- "does not require that the weights of the two branches be shared, not that the architectures be identical, nor that the inputs be of the same nature;
- does not require a memory bank, nor contrastive samples, nor a large batch size;
- does not require batch-wise nor feature-wise normalization; and
- does not require vector quantization nor a predictor module."

Our Study: FairSSL (accepted to ICML 2026)

- Contribution:
  - Extend VICReg to a subject-based setting to obtain fairer predictions

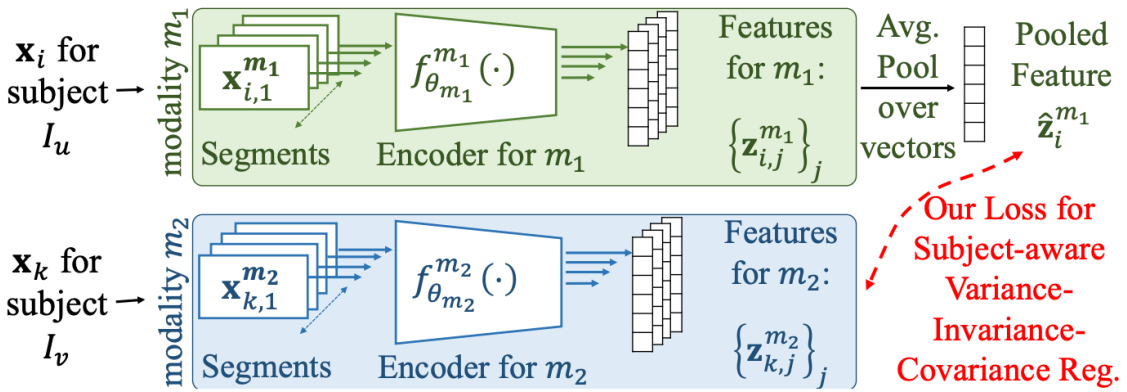
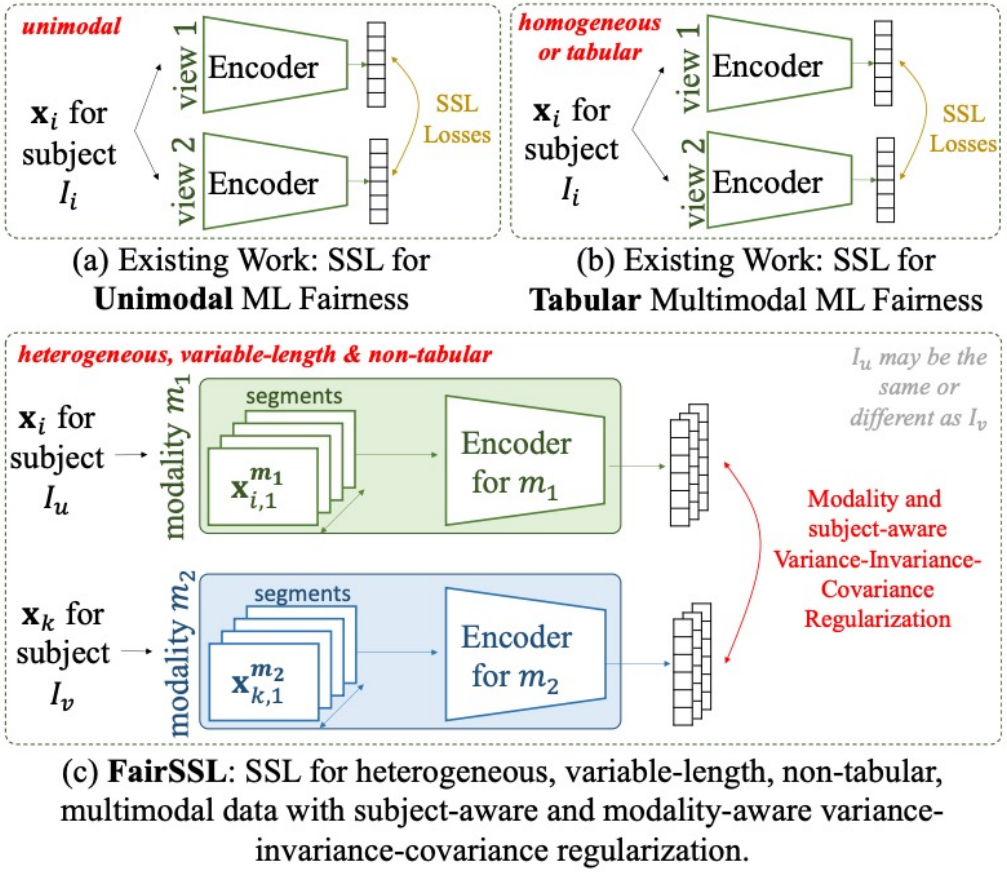


Figure 2. FairSSL processes each modality for the same or different subjects and regularizes representations in a subject-aware manner.



(a) Existing Work: SSL for **Unimodal** ML Fairness

(b) Existing Work: SSL for **Tabular** Multimodal ML Fairness

(c) **FairSSL**: SSL for heterogeneous, variable-length, non-tabular, multimodal data with subject-aware and modality-aware variance-invariance-covariance regularization.

Figure 1. (a,b) Prior work has explored SSL for ML fairness in unimodal or tabular data settings. (c) FairSSL addresses the challenges of non-tabular multimodal data in a subject-aware and modality-aware manner.

JEPA

Assran et al., Self-Supervised Learning from Images with a Joint-Embedding Predictive Architecture, 2023.

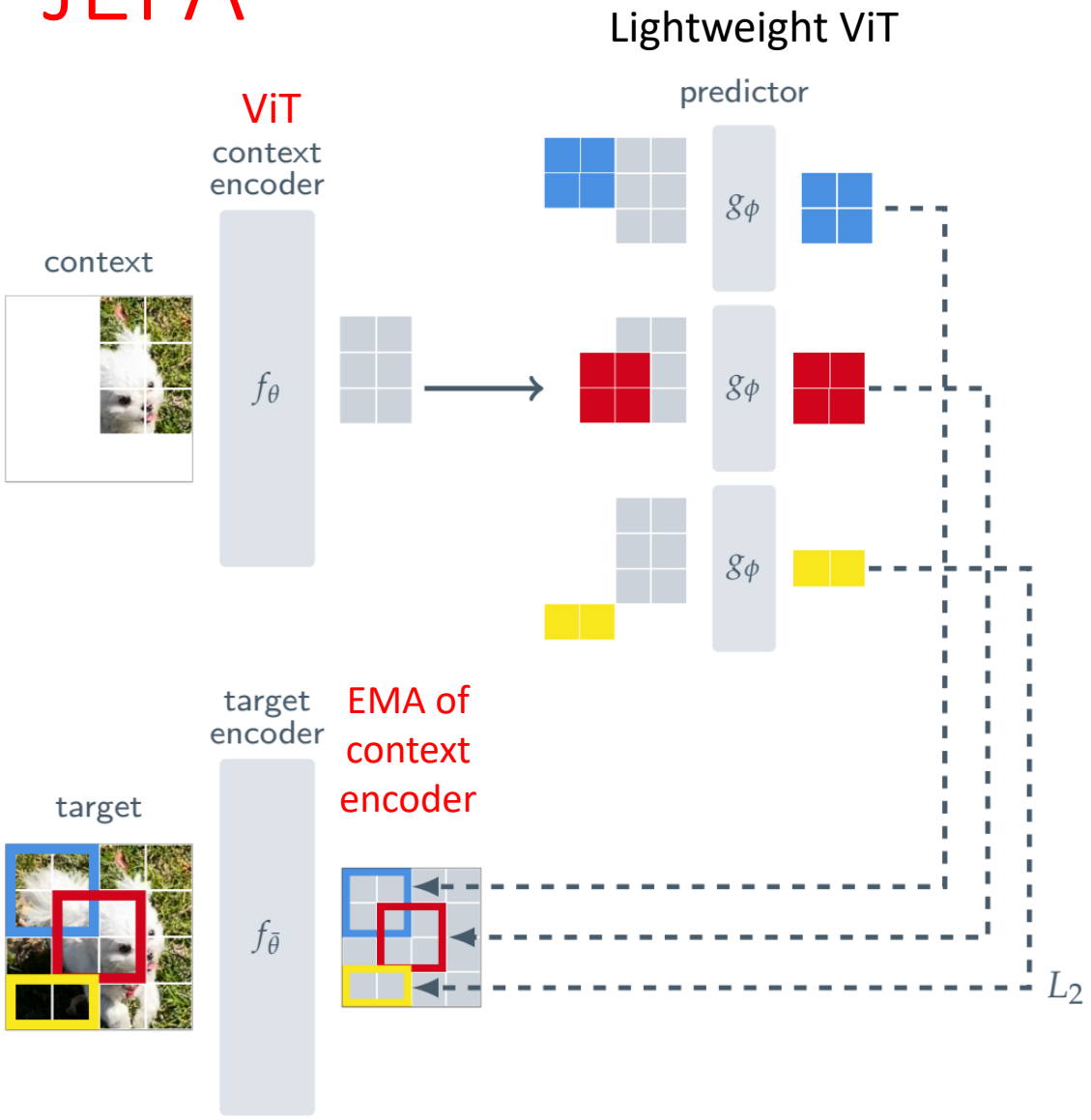


Figure 3. **I-JEPA.** The Image-based Joint-Embedding Predictive Architecture uses a single context block to predict the representations of various target blocks originating from the same image. The context encoder is a Vision Transformer (ViT), which only processes the visible context patches. The predictor is a narrow ViT that takes the context encoder output and, conditioned on positional tokens (shown in color), predicts the representations of a target block at a specific location. The target representations correspond to the outputs of the target-encoder, the weights of which are updated at each iteration via an exponential moving average of the context encoder weights.

LeJEPA

Balestriero, LeJEPA: Provable and Scalable Self-Supervised Learning Without the Heuristics, 2025.

- Approach: Match high-dimensional embedding distribution to an isotropic Gaussian.
- However, this is very expensive and unstable in high dimensions.
- Crame-Wold theorem: For two high-dimensional distributions to be identical, their 1D linear projections should be identical as well.

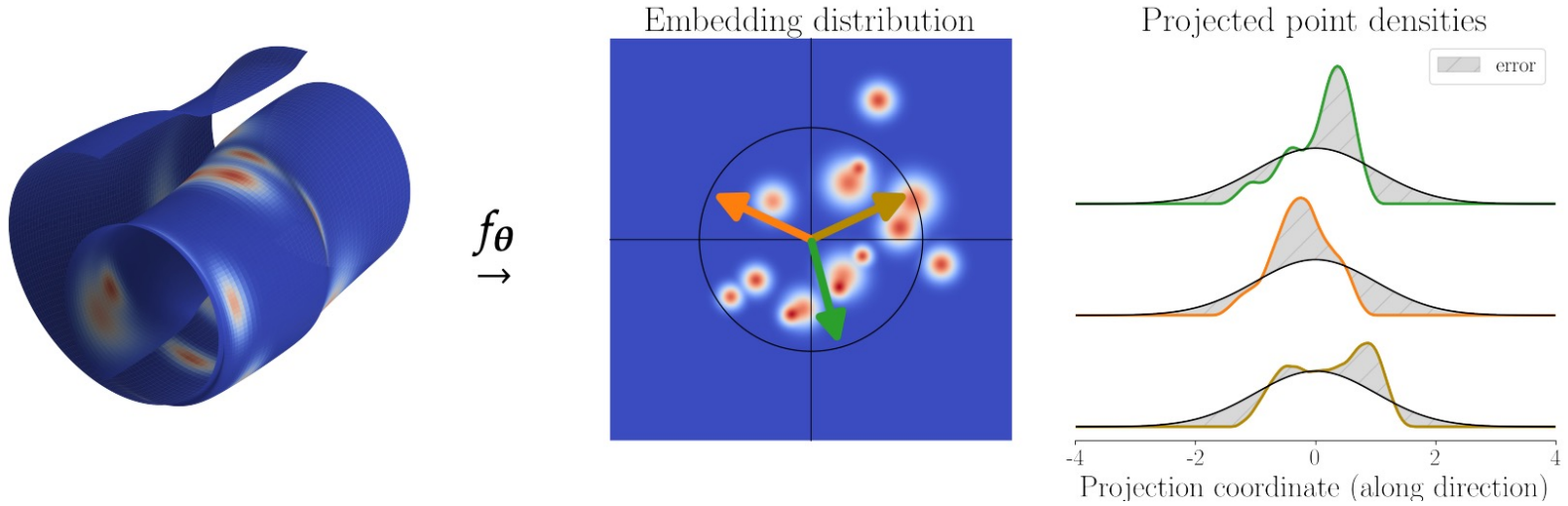


Figure 2. Sketched Isotropic Gaussian Regularization (SIGReg): Given some arbitrary input data with density p_x with support that may or may not lie on a manifold (**left**), a Deep network (DN) encoder (f_θ) produces embeddings $z = f_\theta(x)$ with some distribution $z \sim p_z$ (**middle**). Our proposed Backward Cramér-Wold Statistics (Section 4) objective pushes p_z to match a target distribution p_t by projecting the embeddings along $1d$ directions (**middle, arrows**) and enforcing that the univariate densities (**right, colored lines**) match the distribution of p_t, projected along the same directions. Any popular statistical test (provided in Section 4.2) can assess the goodness-of-fit–in practice we argue for characteristic function tests (Section 4.2). By using SIGReg with p_t isotropic Gaussian (**right, black lines**), we introduce a lean and provably optimal (Section 3) JEPA, coined LeJEPA, free of numerous heuristics and able to produce competitive performances (Sections 5 and 6).

LeJEPA

Balestriero, LeJEPA: Provable and Scalable Self-Supervised Learning Without the Heuristics, 2025.

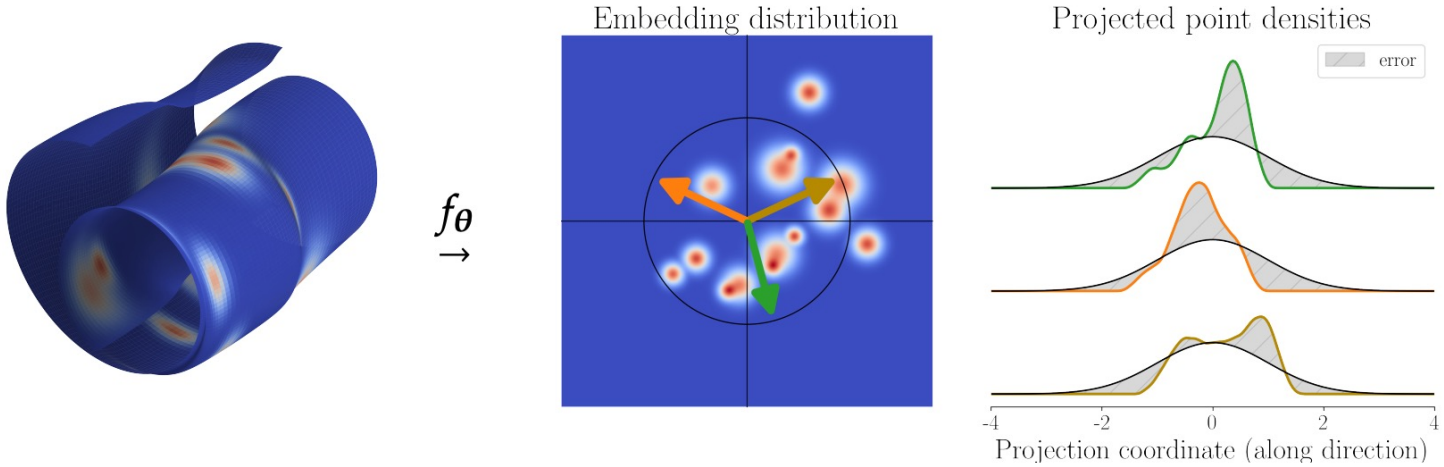


Figure 2. Sketched Isotropic Gaussian Regularization (SIGReg): Given some arbitrary input data with density p_x with support that may or may not lie on a manifold (**left**), a Deep network (DN) encoder (f_θ) produces embeddings $z = f_\theta(x)$ with some distribution $z \sim p_z$ (**middle**). Our proposed Backward Cramér-Wold Statistics (Section 4) objective pushes p_z to match a target distribution p_t by projecting the embeddings along $1d$ directions (**middle, arrows**) and enforcing that the univariate densities (**right, colored lines**) match the distribution of p_t, projected along the same directions. Any popular statistical test (provided in Section 4.2) can assess the goodness-of-fit–in practice we argue for characteristic function tests (Section 4.2). By using SIGReg with p_t isotropic Gaussian (**right, black lines**), we introduce a lean and provably optimal (Section 3) JEPA, coined LeJEPA, free of numerous heuristics and able to produce competitive performances (Sections 5 and 6).

SIGReg sketches a statistical test T towards isotropic Gaussian

$$\text{SIGReg}_T(\mathbb{A}, \{f_\theta(x_n)\}_{n=1}^N) \triangleq \frac{1}{|\mathbb{A}|} \sum_{a \in \mathbb{A}} T(\{a^\top f_\theta(x_n)\}_{n=1}^N),$$

$\mathbb{A}$: The set of random directions.

LeJEPA

Balestriero, LeJEPA: Provable and Scalable Self-Supervised Learning Without the Heuristics, 2025.

Algorithm 2. LeJEPA implementation–works out-of-the-box on any dataset, with DDP, with any backbone, e.g., torchvision or timm. For non-ViT architectures (e.g., ResNet), set global_views = all_views. We use bs for the minibatch size, SIGReg is from algorithm 1.

```
def LeJEPA(global_views, all_views, lambd):
    """global_views and all_views are lists of
        tensors, lambd is a scalar"""

    # embedding of global views
    g_emb = forward(torch.cat(glob_views))
    # embedding of local views
    # if resnet: skip with a_emb=g_emb
    a_emb = forward(torch.cat(all_views))

    # LeJEPA loss
    centers = g_emb.view(-1, bs, K).mean(0)
    a_emb = a_emb.view(-1, bs, K)
    sim = (centers - a_emb).square().mean()
    sigreg = mean(SIGReg(emb, global_step) for emb
        in a_emb)
    return (1-lambd)*sim + lambd*sigreg
```

Non-Contrastive Approaches

Bootstrap Your Own Latent (BYOL – Grill et al., 2020)

- Does not use negative samples. Not contrastive.

Given an image $\mathbf{x}$, the BYOL loss is constructed as follows:

- Create two augmented views: $\mathbf{v} = t(\mathbf{x})$; $\mathbf{v}' = t'(\mathbf{x})$ with augmentations sampled $t \sim \mathcal{T}, t' \sim \mathcal{T}'$;
- Then they are encoded into representations, $\mathbf{y}_\theta = f_\theta(\mathbf{v}), \mathbf{y}' = f_\xi(\mathbf{v}')$;
- Then they are projected into latent variables, $\mathbf{z}_\theta = g_\theta(\mathbf{y}_\theta), \mathbf{z}' = g_\xi(\mathbf{y}')$;
- The online network outputs a prediction $q_\theta(\mathbf{z}_\theta)$;
- Both $q_\theta(\mathbf{z}_\theta)$ and $\mathbf{z}'$ are L2-normalized, giving us $\bar{q}_\theta(\mathbf{z}_\theta) = q_\theta(\mathbf{z}_\theta)/\|q_\theta(\mathbf{z}_\theta)\|$ and $\bar{\mathbf{z}}' = \mathbf{z}'/\|\mathbf{z}'\|$;
- The loss $\mathcal{L}_\theta^{\text{BYOL}}$ is MSE between L2-normalized prediction $\bar{q}_\theta(\mathbf{z})$ and $\bar{\mathbf{z}}'$;
- The other symmetric loss $\tilde{\mathcal{L}}_\theta^{\text{BYOL}}$ can be generated by switching $\mathbf{v}'$ and $\mathbf{v}$; that is, feeding $\mathbf{v}'$ to online network and $\mathbf{v}$ to target network.
- The final loss is $\mathcal{L}_\theta^{\text{BYOL}} + \tilde{\mathcal{L}}_\theta^{\text{BYOL}}$ and only parameters θ are optimized.

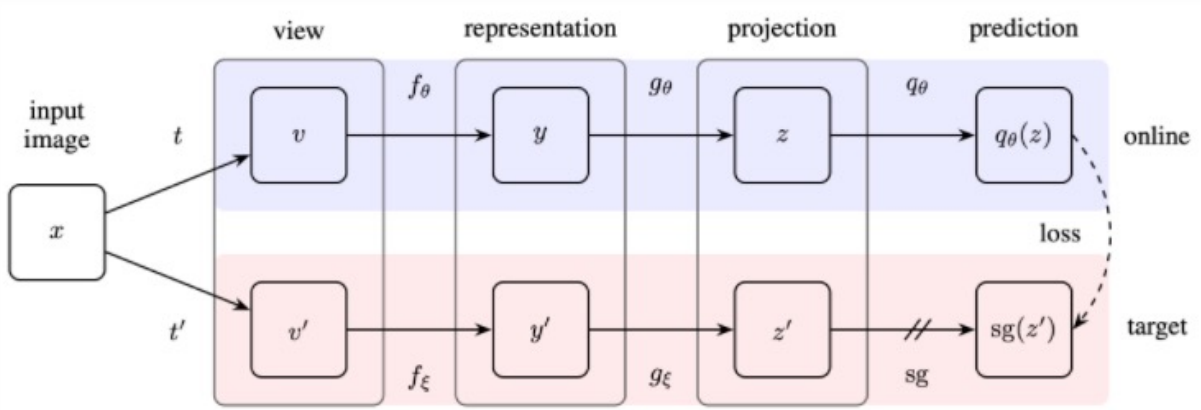


Fig. 10. The model architecture of BYOL. After training, we only care about f_θ for producing representation, $y = f_\theta(x)$, and everything else is discarded. sg means stop gradient. (Image source: Grill, et al 2020)

$$\xi \leftarrow \tau\xi + (1-\tau)\theta.$$

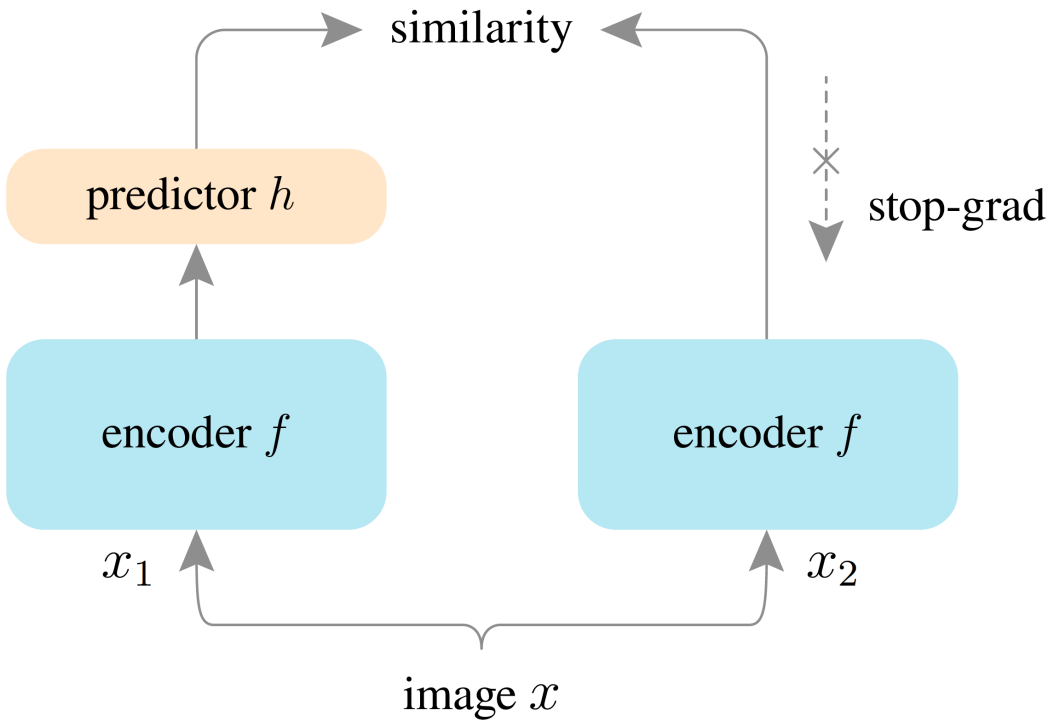
https://lilianweng.github.io/lil-log/2021/05/31/contrastive-representation-learning.html

Simple Siamese Representation Learning (SimSiam – Chen et al., 2020)

- "BYOL without momentum encoder".
- Does not use negative samples. Not contrastive.

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()
```



https://github.com/facebookresearch/simsiam

DINO: Distillation-based

Caron et al., Emerging Properties in Self-Supervised Vision Transformers, 2021.

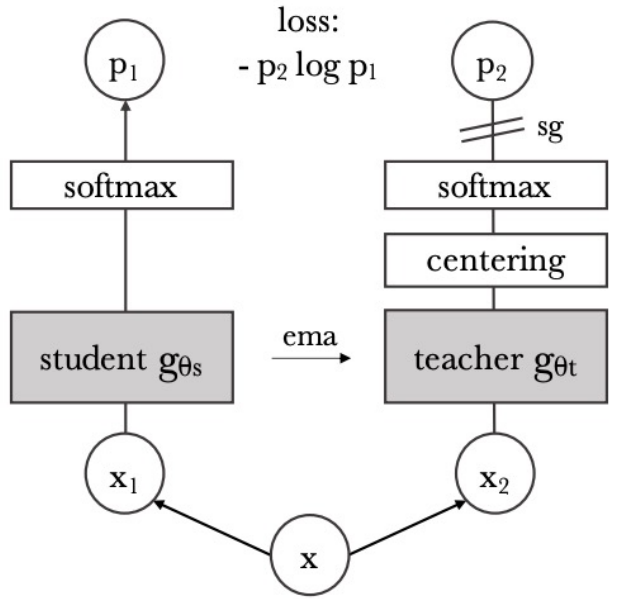


Figure 2: **Self-distillation with no labels.** We illustrate DINO in the case of one single pair of views (x_1, x_2) for simplicity. The model passes two different random transformations of an input image to the student and teacher networks. Both networks have the same architecture but different parameters. The output of the teacher network is centered with a mean computed over the batch. Each networks outputs a K dimensional feature that is normalized with a temperature softmax over the feature dimension. Their similarity is then measured with a cross-entropy loss. We apply a stop-gradient (sg) operator on the teacher to propagate gradients only through the student. The teacher parameters are updated with an exponential moving average (ema) of the student parameters.

Clustering-based Approaches

DeepCluster

Caron et al., Deep Clustering for Unsupervised Learning of Visual Features, 2019.

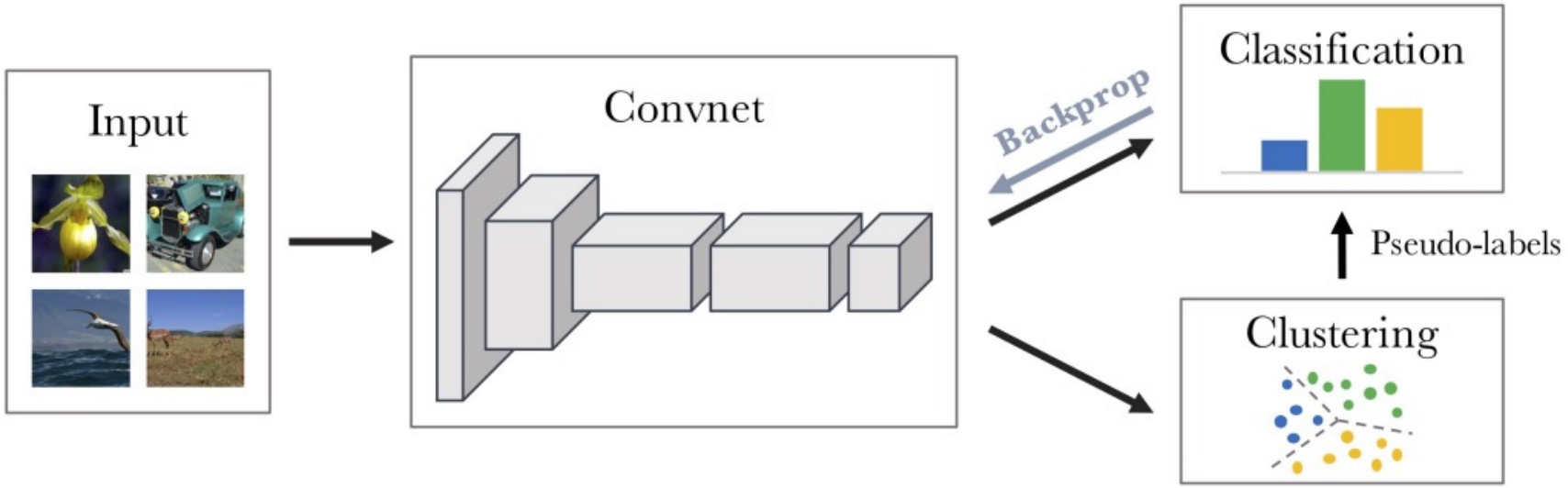


Fig. 1: Illustration of the proposed method: we iteratively cluster deep features and use the cluster assignments as pseudo-labels to learn the parameters of the convnet.

- Starts with a randomly initialized CNN for the clustering part.
- Clustering: K-means clustering.

SwAV

Caron et al., Unsupervised Learning of Visual Features by Contrasting Cluster Assignments, 2021.

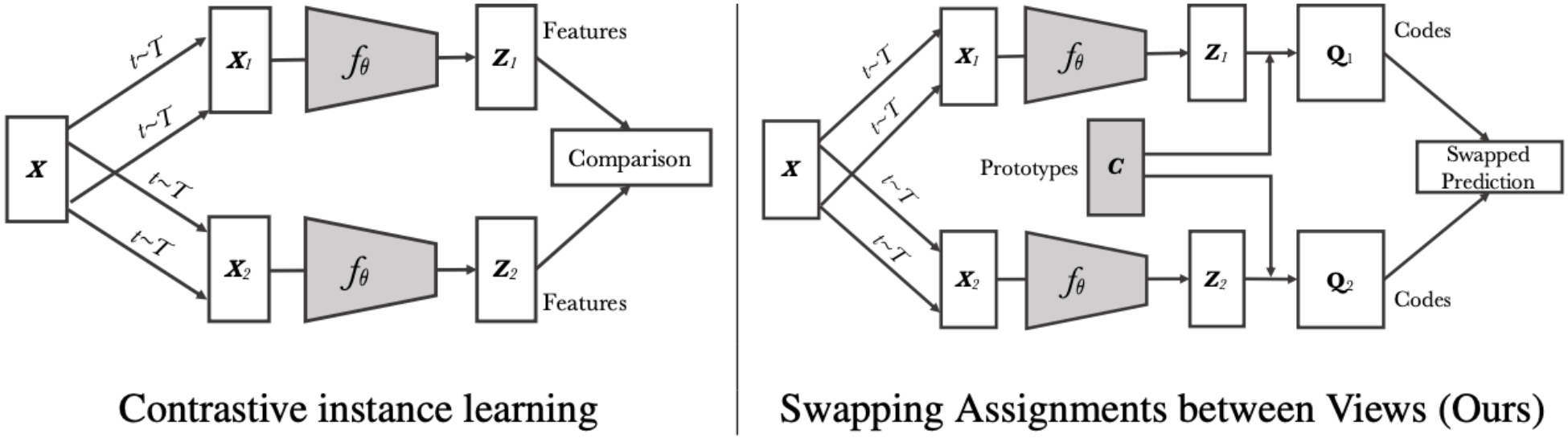
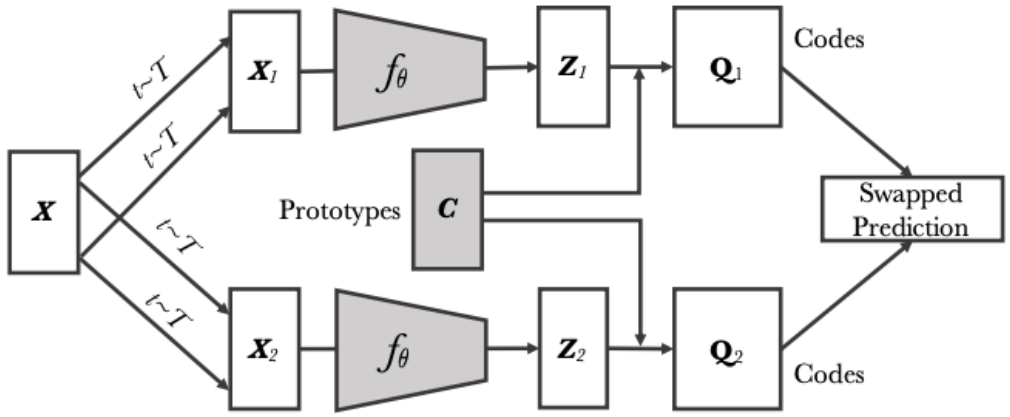


Figure 1: **Contrastive instance learning (left) *vs*. SwAV (right).** In contrastive learning methods applied to instance classification, the features from different transformations of the same images are compared directly to each other. In SwAV, we first obtain "codes" by assigning features to prototype vectors. We then solve a "swapped" prediction problem wherein the codes obtained from one data augmented view are predicted using the other view. Thus, SwAV does not directly compare image features. Prototype vectors are learned along with the ConvNet parameters by backpropragation.

SwAV

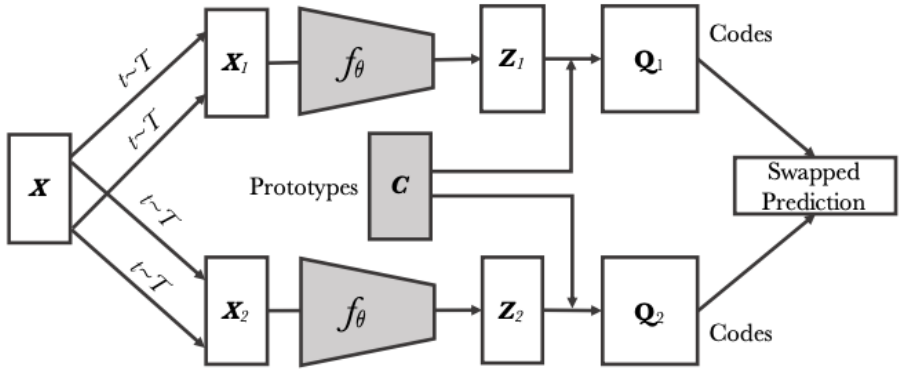
In contrast to DeepCluster,

- SwAV uses online clustering (creating prototypes).
- SwAV uses soft-assignment to clusters (optimal transport).
- SwAV uses augmentations.



Swapping Assignments between Views (Ours)

SwAV



Swapping Assignments between Views (Ours)

More precisely, we compute a code from an augmented version of the image and predict this code from other augmented versions of the same image. Given two image features $\mathbf{z}_t$ and $\mathbf{z}_s$ from two different augmentations of the same image, we compute their codes $\mathbf{q}_t$ and $\mathbf{q}_s$ by matching these features to a set of K prototypes $\{\mathbf{c}_1, \ldots, \mathbf{c}_K\}$. We then setup a "swapped" prediction problem with the following loss function:

$$L(\mathbf{z}_t, \mathbf{z}_s) \quad = \quad \ell(\mathbf{z}_t, \mathbf{q}_s) + \ell(\mathbf{z}_s, \mathbf{q}_t), \tag{1}$$

where the function $\ell(\mathbf{z}, \mathbf{q})$ measures the fit between features $\mathbf{z}$ and a code $\mathbf{q}$, as detailed later.

$$\ell(\mathbf{z}_t, \mathbf{q}_s) = -\sum_k \mathbf{q}_s^{(k)} \log \mathbf{p}_t^{(k)}, \quad \text{where} \quad \mathbf{p}_t^{(k)} = \frac{\exp\left(\frac{1}{\tau}\mathbf{z}_t^\top \mathbf{c}_k\right)}{\sum_{k'} \exp\left(\frac{1}{\tau}\mathbf{z}_t^\top \mathbf{c}_{k'}\right)}.$$

SwAV

- Prototypes are learnable embeddings and updated during training.
- Balanced optimal transport objective is used to ensure that samples are equally distributed to the prototypes.

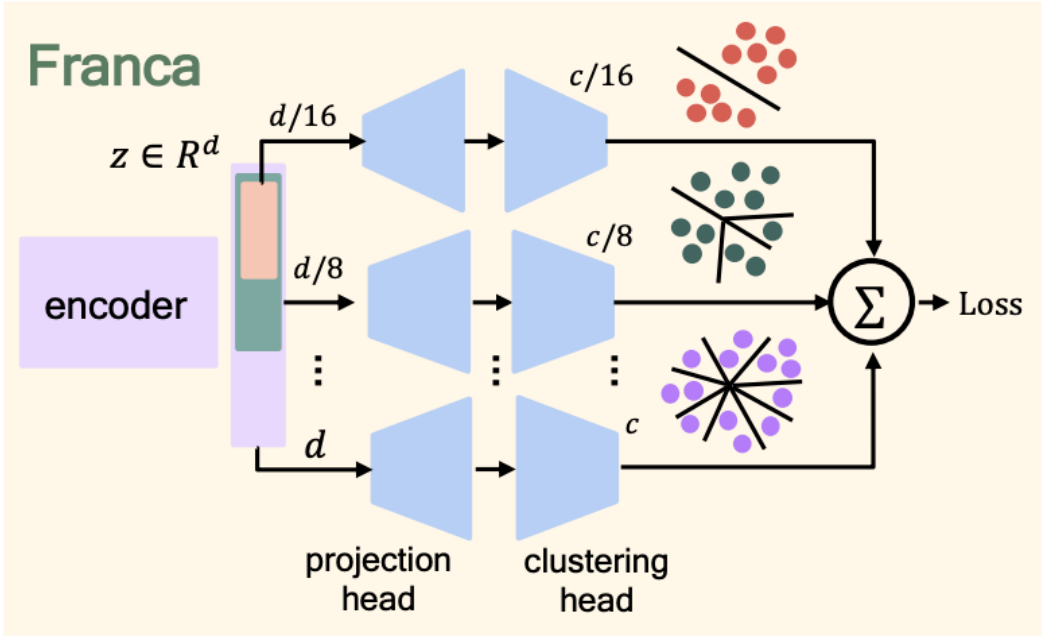
Each image $\mathbf{x}_n$ is transformed into an augmented view $\mathbf{x}_{nt}$ by applying a transformation t sampled from the set $\mathcal{T}$ of image transformations. The augmented view is mapped to a vector representation by applying a non-linear mapping f_θ to $\mathbf{x}_{nt}$. The feature is then projected to the unit sphere, *i.e.*, $\mathbf{z}_{nt} = f_\theta(\mathbf{x}_{nt})/\|f_\theta(\mathbf{x}_{nt})\|_2$. We then compute a code $\mathbf{q}_{nt}$ from this feature by mapping $\mathbf{z}_{nt}$ to a set of K trainable prototypes vectors, $\{\mathbf{c}_1, \ldots, \mathbf{c}_K\}$. We denote by $\mathbf{C}$ the matrix whose columns are the $\mathbf{c}_1, \ldots, \mathbf{c}_k$. We now describe how to compute these codes and update the prototypes online.

Venkataramanan et al., Franca: Nested Matryoshka Clustering for Scalable Visual Representation Learning, 2025.

Franca

We present Franca (pronounced Fran-ka): ‘free’ one; the first fully open-source (data, code, weights) vision foundation model that matches—and in many cases surpasses—the performance of state-of-the-art proprietary models, e.g., DINOv2, CLIP, SigLIPv2, etc. Our approach is grounded in a transparent training pipeline inspired from WebSSL and uses publicly available data: Imagenet-21K and LAION-COCO (600M images). Beyond model release, we tackle critical limitations in self-supervised learning clustering methods. Existing approaches assign image features to large codebooks via clustering algorithms such as Sinkhorn-Knopp, but they often overlook the inherent ambiguity in cluster semantics. To address this, we introduce a multi-head clustering projector based on nested Matryoshka representations. This design progressively refines features into increasingly fine-grained clusters without increasing the model size, producing higher-quality dense representations. Additionally, we propose a novel positional disentanglement strategy that explicitly removes positional biases from dense representations. This leads to consistent gains on several downstream tasks, demonstrating the utility of cleaner feature spaces.

Method



Resources on SSL

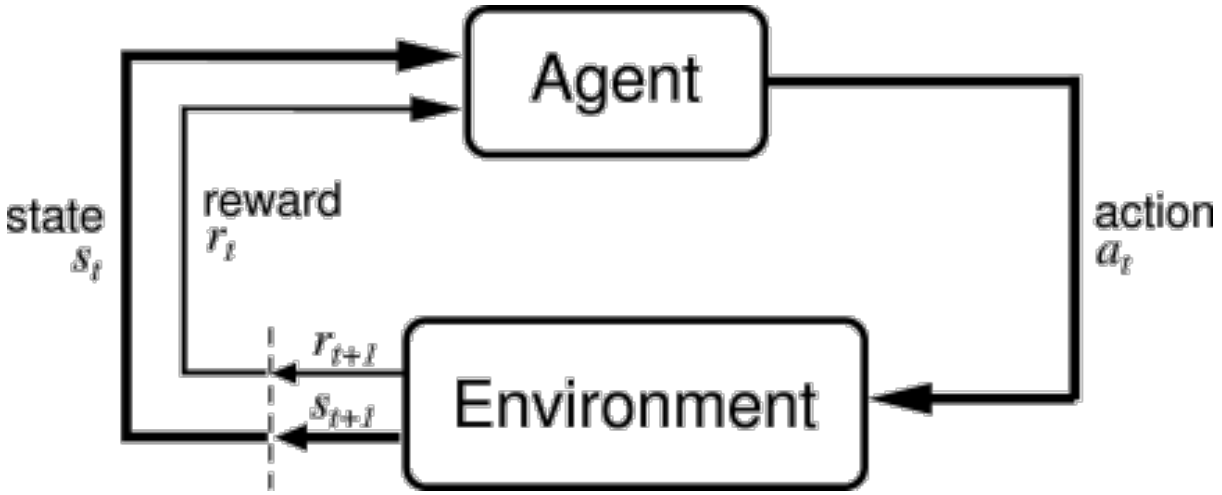
- The rise of SSL, by Y. Lecun: https://www.youtube.com/watch?v=05wUrb5Ej8Q&t=21252s

- Self-supervised representation learning: https://lilianweng.github.io/lil-log/2019/11/10/self-supervised-learning.html

- Kumar et al., Contrastive self-supervised learning: review, progress, challenges and future research directions, 2022.

Deep reinforcement learning

Reinforcement Learning



The agent receives reward r_t for its actions.

More formally

- An agent's behavior is defined by a policy, π:

$$\pi: \mathcal{S} \rightarrow P(\mathcal{A})$$

$\mathcal{S}$: The space of states.

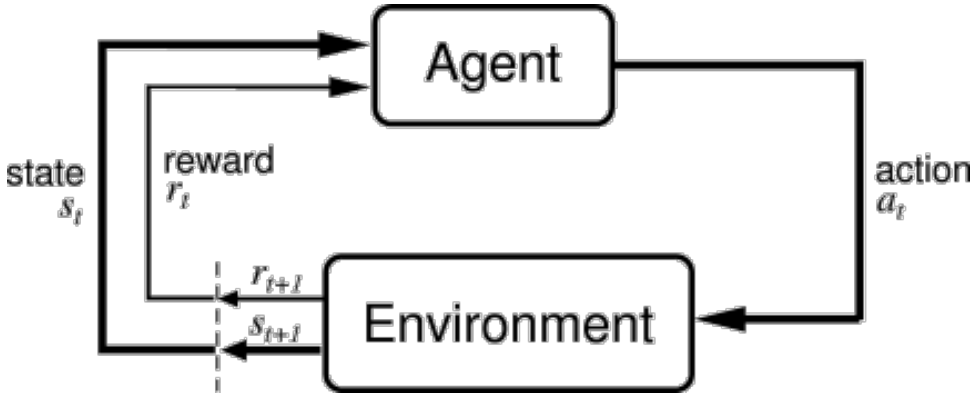
$\mathcal{A}$: The space of actions.

- The "return" from a state is usually:

$$R_t = \sum_{i=t}^{T} \gamma^{(i-t)} r(s_i, a_i)$$

$r(s_i, a_i)$: the reward for action a_i in state s_i.

γ: discount factor.



- Goal: Learn a policy that maximizes the expected return from the starting position:

$$\mathbb{E}_{r_i, s_i \sim E, a_i \sim \pi}[R_1]$$

http://www.cs.ubc.ca/~murphyk/Bayes/pomdp.html

More formally

- We can define an expected return for taking action a_t at state s_t:

$$Q^{\pi}(s_t, a_t) = \mathbb{E}_{r_{i \geq t}, s_{i>t} \sim E, a_{i>t} \sim \pi} \left[R_t \mid s_t, a_t \right]$$

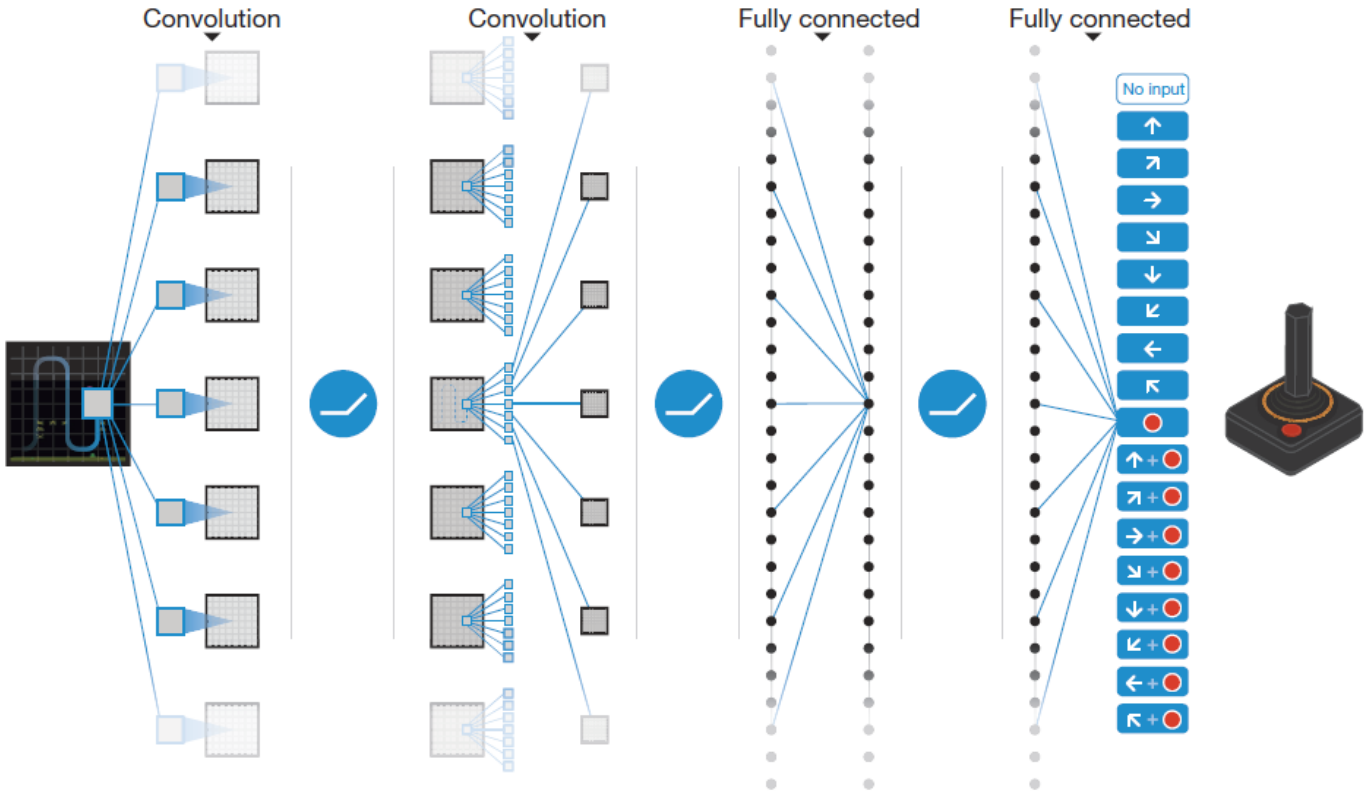
- This can be rewritten as (the so-called the Bellman equation):

$$Q^{\pi}(s_t, a_t) = \mathbb{E}_{r_t, s_{t+1} \sim E} \left[r(s_t, a_t) + \gamma \, \mathbb{E}_{a_{t+1} \sim \pi} [Q^{\pi}(s_{t+1}, a_{t+1})] \right]$$

http://www.cs.ubc.ca/~murphyk/Bayes/pomdp.html

Reinforcement Learning with Deep Networks

- Two general approaches:
  - Value gradients
  - Policy gradients



Q values of actions are predicted at the output.

Figure 1 | Schematic illustration of the convolutional neural network. The details of the architecture are explained in the Methods. The input to the neural network consists of an $84 \times 84 \times 4$ image produced by the preprocessing map ϕ, followed by three convolutional layers (note: snaking blue line symbolizes sliding of each filter across input image) and two fully connected layers with a single output for each valid action. Each hidden layer is followed by a rectifier nonlinearity (that is, $\max(0,x)$).

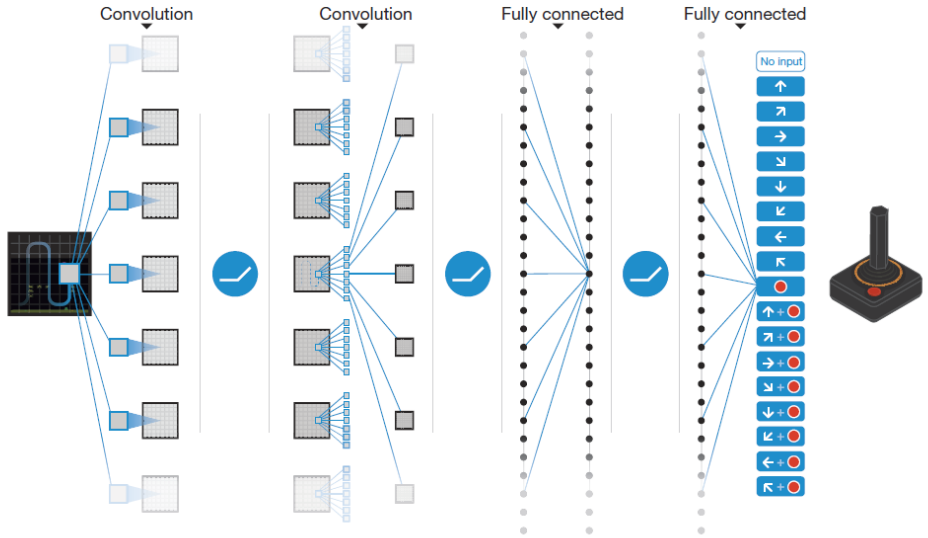
LETTER

2015

doi:10.1038/nature14236

Human-level control through deep reinforcement learning

Volodymyr Mnih[1]*, Koray Kavukcuoglu[1]*, David Silver[1]*, Andrei A. Rusu[1], Joel Veness[1], Marc G. Bellemare[1], Alex Graves[1], Martin Riedmiller[1], Andreas K. Fidjeland[1], Georg Ostrovski[1], Stig Petersen[1], Charles Beattie[1], Amir Sadik[1], Ioannis Antonoglou[1], Helen King[1], Dharshan Kumaran[1], Daan Wierstra[1], Shane Legg[1] & Demis Hassabis[1]



network. We refer to a neural network function approximator with weights θ as a Q-network. A Q-network can be trained by adjusting the parameters θ_i at iteration i to reduce the mean-squared error in the Bellman equation, where the optimal target values $r+\gamma \max_{a'} Q^*(s',a')$ are substituted with approximate target values $y=r+\gamma \max_{a'} Q(s',a';\theta_i^-)$, using parameters θ_i^- from some previous iteration. This leads to a sequence of loss functions $L_i(\theta_i)$ that changes at each iteration i,

$$L_i(\theta_i)=\mathbb{E}_{s,a,r}\left[(\mathbb{E}_{s'}[y|s,a]-Q(s,a;\theta_i))^2\right]$$

Algorithm 1: deep Q-learning with experience replay.

Initialize replay memory D to capacity N

Initialize action-value function Q with random weights θ

Initialize target action-value function $\hat{Q}$ with weights $\theta^{-} = \theta$

For episode $= 1, M$ **do**

  Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$

  For $t = 1,\mathrm{T}$ **do**

    With probability ε select a random action a_t

    otherwise select $a_t = \mathrm{argmax}_a Q(\phi(s_t), a; \theta)$

    Execute action a_t in emulator and observe reward r_t and image x_{t+1}

    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$

    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D

    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D

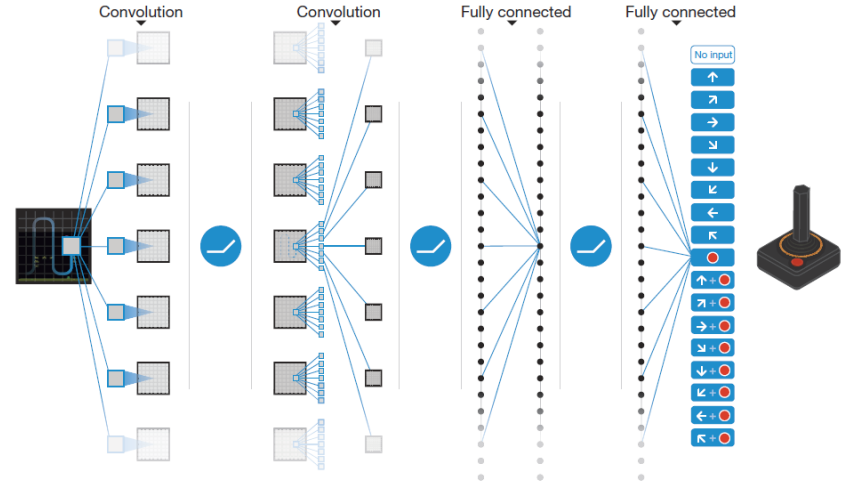
    Set $y_j = \begin{cases} r_j & \text{if episode terminates at step } j+1 \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^{-}) & \text{otherwise} \end{cases}$

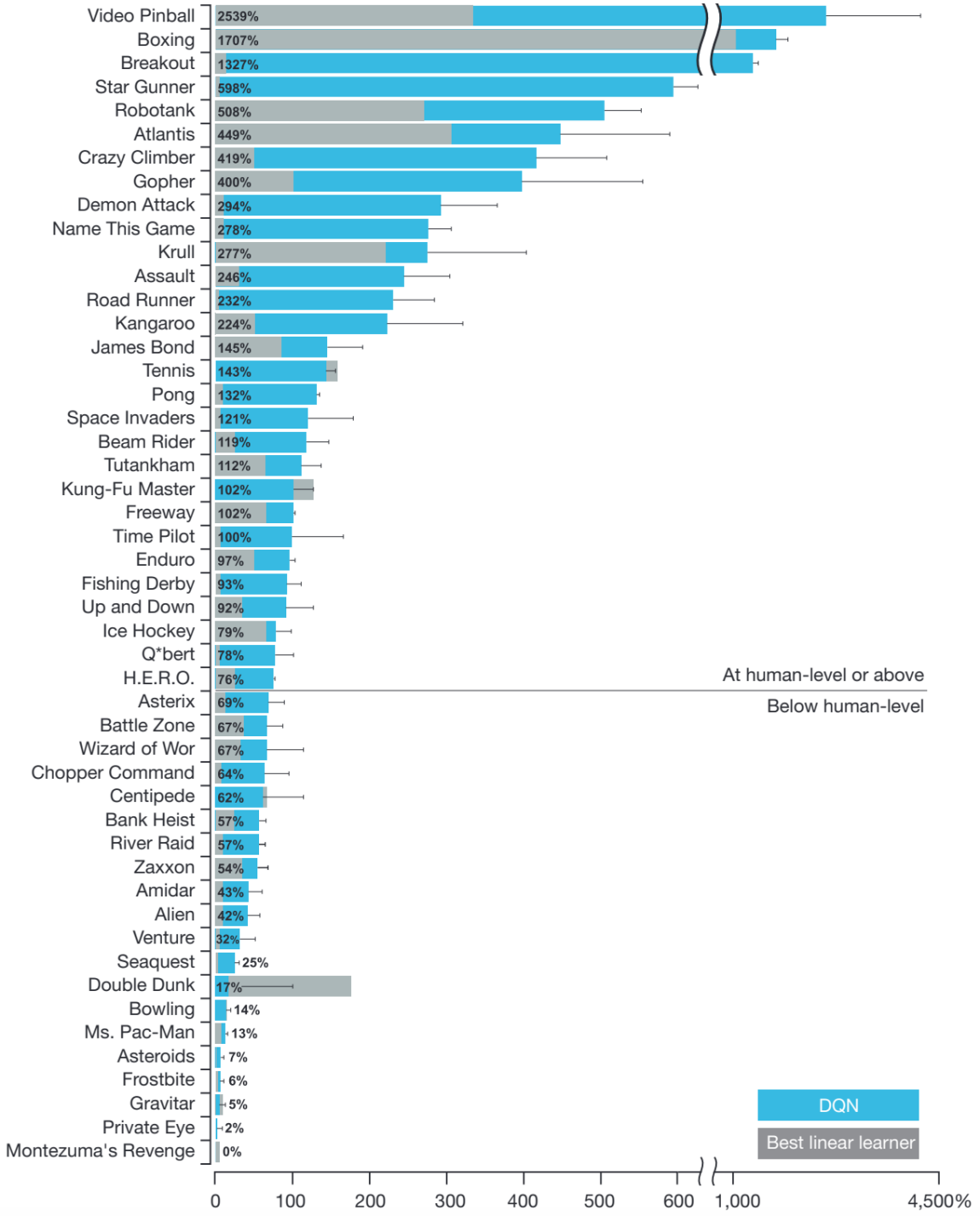
    Perform a gradient descent step on $\left(y_j - Q(\phi_j, a_j; \theta)\right)^2$ with respect to the network parameters θ

    Every C steps reset $\hat{Q} = Q$

  End For

End For





LETTER
doi:10.1038/nature14236

Human-level control through deep reinforcement learning

Volodymyr Mnih[1]*, Koray Kavukcuoglu[1]*, David Silver[1]*, Andrei A. Rusu[1], Joel Veness[1], Marc G. Bellemare[1], Alex Graves[1], Martin Riedmiller[1], Andreas K. Fidjeland[1], Georg Ostrovski[1], Stig Petersen[1], Charles Beattie[1], Amir Sadik[1], Ioannis Antonoglou[1], Helen King[1], Dharshan Kumaran[1], Daan Wierstra[1], Shane Legg[1] & Demis Hassabis[1]

Double DQN

Deep Reinforcement Learning with Double Q-learning

Hado van Hasselt and **Arthur Guez** and **David Silver** 2015
Google DeepMind

Problem with DQN (and Q learning):

- Over-optimistic estimation owing to the max because the environment is noisy

Q-learning

$Q(s, a; \boldsymbol{\theta}_t)$. The standard Q-learning update for the parameters after taking action A_t in state S_t and observing the immediate reward R_{t+1} and resulting state S_{t+1} is then

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha(Y_t^{\mathrm{Q}} - Q(S_t, A_t; \boldsymbol{\theta}_t)) \nabla_{\boldsymbol{\theta}_t} Q(S_t, A_t; \boldsymbol{\theta}_t) . \quad (1)$$

where α is a scalar step size and the target Y_t^{Q} is defined as

$$Y_t^{\mathrm{Q}} \equiv R_{t+1} + \gamma \max_a Q(S_{t+1}, a; \boldsymbol{\theta}_t) . \quad (2)$$

This update resembles stochastic gradient descent, updating the current value $Q(S_t, A_t; \boldsymbol{\theta}_t)$ towards a target value Y_t^{Q}.

DQN

work, and the use of experience replay. The target network, with parameters $\boldsymbol{\theta}^-$, is the same as the online network except that its parameters are copied every τ steps from the online network, so that then $\boldsymbol{\theta}_t^- = \boldsymbol{\theta}_t$, and kept fixed on all other steps. The target used by DQN is then

$$Y_t^{\mathrm{DQN}} \equiv R_{t+1} + \gamma \max_a Q(S_{t+1}, a; \boldsymbol{\theta}_t^-) . \quad (3)$$

Double DQN

Deep Reinforcement Learning with Double Q-learning

Hado van Hasselt and **Arthur Guez** and **David Silver** 2015
Google DeepMind

Solution

- Separate action selection (actor) from action evaluation (critic)

Double Q-learning

The Double Q-learning error can then be written as

$$Y_t^{\text{DoubleQ}} \equiv R_{t+1} + \gamma Q(S_{t+1}, \operatorname*{argmax}_a Q(S_{t+1}, a; \boldsymbol{\theta}_t); \boldsymbol{\theta}'_t) .$$

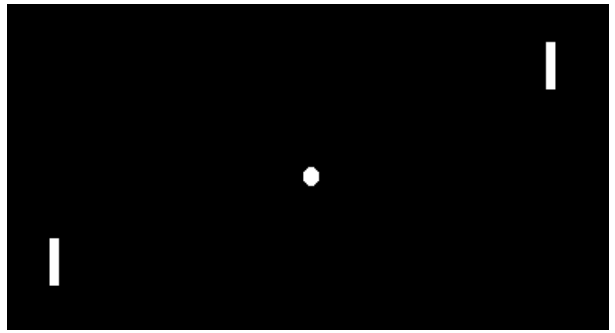
Double DQN (DDQN)

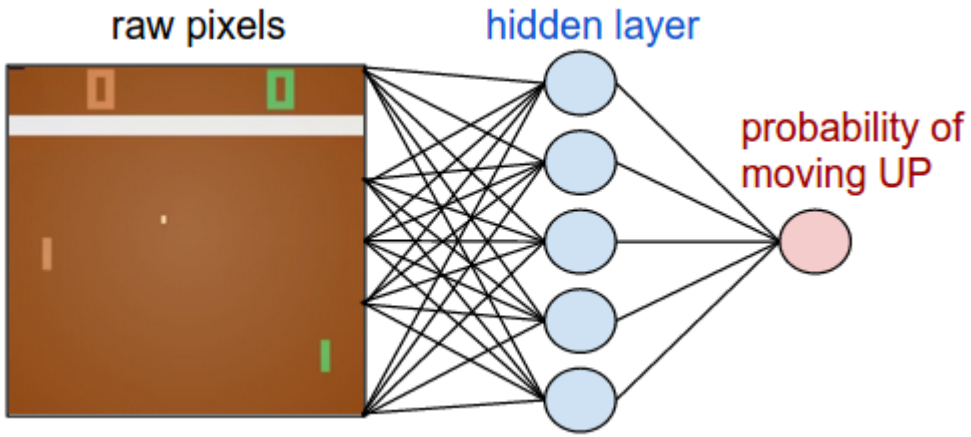
to the resulting algorithm as Double DQN. Its update is the same as for DQN, but replacing the target Y_t^{DQN} with

$$Y_t^{\text{DoubleDQN}} \equiv R_{t+1} + \gamma Q(S_{t+1}, \operatorname*{argmax}_a Q(S_{t+1}, a; \boldsymbol{\theta}_t), \boldsymbol{\theta}_t^-) .$$

In comparison to Double Q-learning (4), the weights of the second network $\boldsymbol{\theta}'_t$ are replaced with the weights of the target network $\boldsymbol{\theta}_t^-$ for the evaluation of the current greedy policy. The update to the target network stays unchanged from DQN, and remains a periodic copy of the online network.

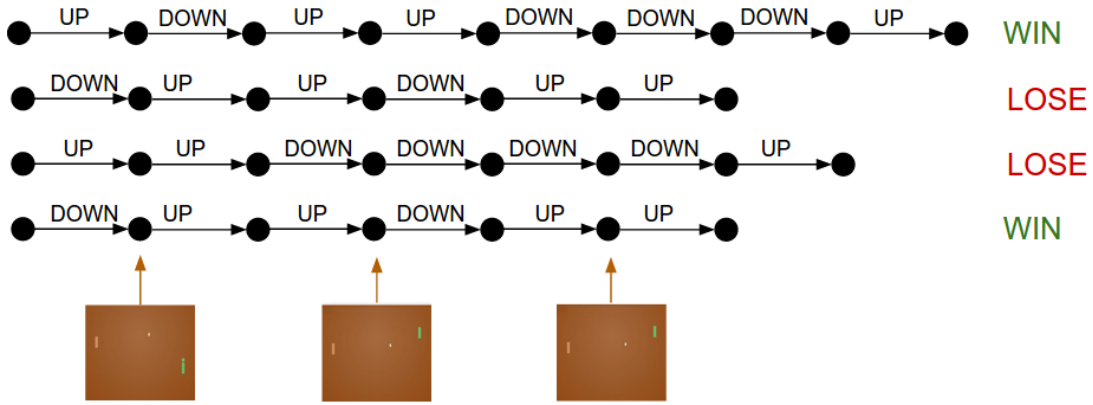
Policy gradients

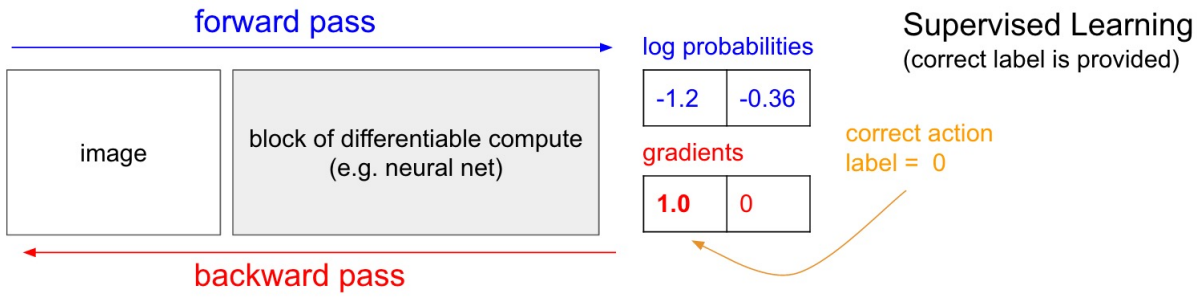




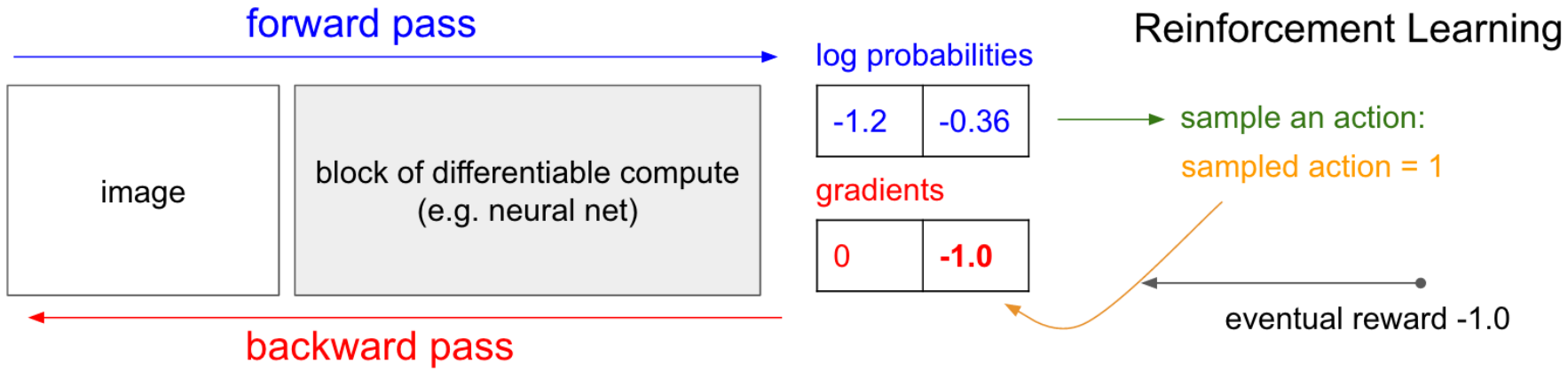
http://karpathy.github.io/2016/05/31/rl/

Policy gradients





$$\nabla_W \log p(y = \text{UP} \mid x)$$



http://karpathy.github.io/2016/05/31/rl/

Policy gradients

Let us start with the defined objective function $J(\theta)$. We can expand the expectation as:

$$J(\theta) = \mathbb{E}[\sum_{t=0}^{T-1} r_{t+1} | \pi_\theta]$$
$$= \sum_{t=i}^{T-1} P(s_t, a_t | \tau) r_{t+1}$$

where i is an arbitrary starting point in a trajectory, $P(s_t, a_t|\tau)$ is the probability of the occurrence of s_t, a_t given the trajectory τ.

https://medium.com/@thechrisyoon/deriving-policy-gradients-and-implementing-reinforce-f887949bd63

Policy gradients

Differentiate both sides with respect to policy parameter θ:

$$\text{Using} \quad \frac{d}{dx} log f(x) = \frac{f'(x)}{f(x)},$$

$$\begin{aligned}
\nabla_\theta J(\theta) &= \sum_{t=i}^{T-1} \nabla_\theta P(s_t, a_t|\tau) r_{t+1} \\
&= \sum_{t=i}^{T-1} P(s_t, a_t|\tau) \frac{\nabla_\theta P(s_t, a_t|\tau)}{P(s_t, a_t|\tau)} r_{t+1} \\
&= \sum_{t=i}^{T-1} P(s_t, a_t|\tau) \nabla_\theta log P(s_t, a_t|\tau) r_{t+1} \\
&= \mathbb{E}[\sum_{t=i}^{T-1} \nabla_\theta log P(s_t, a_t|\tau) r_{t+1}]
\end{aligned}$$

This however does not depend on the policy network

https://medium.com/@thechrisyoon/deriving-policy-gradients-and-implementing-reinforce-f887949bd63

Policy gradients

By rewriting the probability as:

$$\begin{aligned} P(s_t, a_t|\tau) &= P(s_0, a_0, s_1, a_2, ..., s_{t-1}, a_{t-1}, s_t, a_t|\pi_\theta) \\ &= P(s_0)\pi_\theta(a_1|s_0)P(s_1|s_0, a_0)\pi_\theta(a_2|s_1)P(s_2|s_1, a_1)\pi_\theta(a_3|s_2) \\ &\quad ...P(s_{t-1}|s_{t-2}, a_{t-2})\pi_\theta(a_{t-1}|s_{t-2})P(s_t|s_{t-1}, a_{t-1})\pi_\theta(a_t|s_{t-1}) \end{aligned}$$

Taking the logarithm and the derivative:

$$\begin{aligned} \nabla_\theta log P(s_t, a_t|\tau) &= 0 + \nabla_\theta log\pi_\theta(a_1|s_0) + 0 + \nabla_\theta log\pi_\theta(a_2|s_1) + 0 + \nabla_\theta log\pi_\theta(a_3|s_2) + \\ &\quad ... + 0 + \nabla_\theta log\pi_\theta(a_{t-1}|s_{t-2}) + 0 \\ &= \nabla_\theta log\pi_\theta(a_1|s_0) + \nabla_\theta log\pi_\theta(a_2|s_1) + \nabla_\theta log\pi_\theta(a_3|s_2) + \\ &\quad ... + \nabla_\theta log\pi_\theta(a_{t-1}|s_{t-2}) + log\pi_\theta(a_t|s_{t-1}) \\ &= \sum_{t'=0}^{t} \nabla_\theta log\pi_\theta(a_{t'}|s_{t'}) \end{aligned}$$

https://medium.com/@thechrisyoon/deriving-policy-gradients-and-implementing-reinforce-f887949bd63

Policy gradients

Incorporating the discount factor $\gamma \in [0, 1]$ into our objective (in order to weight immediate rewards more than future rewards):

$$J(\theta) = \mathbb{E}[\gamma^0 r_1 + \gamma^1 r_2 + \gamma^2 r_3 + ... + \gamma^{T-1} r_T | \pi_\theta]$$

We can perform a similar derivation to obtain

$$\nabla_\theta J(\theta) = \sum_{t=0}^{T-1} \nabla_\theta log \pi_\theta(a_t|s_t)(\sum_{t'=t+1}^{T} \gamma^{t'-t-1} r_{t'})$$

and simplifying $\sum_{t'=t+1}^{T} \gamma^{t'-t-1} r_{t'}$ to G_t,

$$\nabla_\theta J(\theta) = \sum_{t=0}^{T-1} \nabla_\theta log \pi_\theta(a_t|s_t) G_t$$

https://medium.com/@thechrisyoon/deriving-policy-gradients-and-implementing-reinforce-f887949bd63

Actor-Critic Networks

Two main components in policy gradient are the policy model and the value function. It makes a lot of sense to learn the value function in addition to the policy, since knowing the value function can assist the policy update, such as by reducing gradient variance in vanilla policy gradients, and that is exactly what the **Actor-Critic** method does.

Actor-critic methods consist of two models, which may optionally share parameters:

- **Critic** updates the value function parameters w and depending on the algorithm it could be action-value $Q_w(a|s)$ or state-value $V_w(s)$.
- **Actor** updates the policy parameters θ for $\pi_\theta(a|s)$, in the direction suggested by the critic.

https://lilianweng.github.io/posts/2018-04-08-policy-gradient/

Actor-Critic Networks

1. Initialize s, θ, w at random; sample $a \sim \pi_\theta(a|s)$.
2. For $t = 1 \dots T$:
    1. Sample reward $r_t \sim R(s, a)$ and next state $s' \sim P(s'|s, a)$;
    2. Then sample the next action $a' \sim \pi_\theta(a'|s')$;
    3. Update the policy parameters: $\theta \leftarrow \theta + \alpha_\theta Q_w(s, a) \nabla_\theta \ln \pi_\theta(a|s)$;
    4. Compute the correction (TD error) for action-value at time t:
    $\delta_t = r_t + \gamma Q_w(s', a') - Q_w(s, a)$
    and use it to update the parameters of action-value function:
    $w \leftarrow w + \alpha_w \delta_t \nabla_w Q_w(s, a)$
    5. Update $a \leftarrow a'$ and $s \leftarrow s'$.

https://lilianweng.github.io/posts/2018-04-08-policy-gradient/

Actor-Critic Networks

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a)\, G_t\right] \quad \text{REINFORCE}$$

$$= \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a)\; Q^w(s,a)\right] \quad \text{Q Actor-Critic}$$

$$= \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a)\; A^w(s,a)\right] \quad \text{Advantage Actor-Critic}$$

$$= \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a)\; \delta\right] \quad \text{TD Actor-Critic}$$

From CMU CS10703 lecture slides

Introducing baseline $b(s)$:

$$\nabla_\theta J(\theta) = \mathbb{E}\left[\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t)(G_t - b(s_t))\right]$$

https://medium.com/@thechrisyoon/deriving-policy-gradients-and-implementing-reinforce-f887949bd63

Actor-Critic Networks

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a)\, G_t\right] \quad \text{REINFORCE}$$

$$= \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a)\; Q^w(s,a)\right] \quad \text{Q Actor-Critic}$$

$$= \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a)\; A^w(s,a)\right] \quad \text{Advantage Actor-Critic}$$

$$= \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a)\; \delta\right] \quad \text{TD Actor-Critic}$$

From CMU CS10703 lecture slides

$$A(s_t, a_t) = Q_w(s_t, a_t) - V_v(s_t)$$

$$= r_{t+1} + \gamma V_v(s_{t+1}) - V_v(s_t)$$

$$V^\pi(s) = E_{a\sim\pi}\left\{\sum_{t=0}^{\infty} \gamma^t R_{t+1} \middle| S_0 = s\right\}$$

https://medium.com/@thechrisyoon/deriving-policy-gradients-and-implementing-reinforce-f887949bd63

Model-based vs. Model-free

- Model-free
  - Learn the Q values
- Model-based
  - Learn the Q values and the transition probabilities (the model of how the environment would change)

RL Methods	Advantages	Disadvantages
Model-based RL	– Small number of interactions between robot & environment – Faster convergence to optimal solution.	–Depend on transition models – Model accuracy has a big impact on learning tasks
Model-free RL	– No need for prior knowledge of transitions – Easily implementable	– Slow learning convergence – High wear & tear of the robot – High risk of damage

Table & Fig: https://www.chenshiyu.top/blog/2019/06/12/An-Overview-of-Model-Based-Reinforcement-Learning/

On-policy vs. Off-policy

This brings us to the key difference between on-policy and off-policy learning: ***On-policy algorithms attempt to improve upon the current behavior policy that is used to make decisions and therefore these algorithms learn the value of the policy carried out by the agent,*** Q^{π}***. Off-policy algorithms learn the value of the optimal policy,*** Q^{*}***, and can improve upon a policy that is different from the behavior policy.*** Determining if the update and behavior policy are the same or different can give us insight into whether or not the algorithm is on-policy or off-policy. If the update policy and the behavior policy are the same, then this suggest but does not guarantee that the learning method is on-policy. If they are different, this suggests that the learning method is off-policy.

	On-Policy	Off-Policy
Advantages	• Learns safer strategy • Often converges faster • Often has better online performance	• More likely to find optimal policy • Less likely to get stuck in local minimum • Can utilize experience replay • Data can be collected via various method
Disadvantages	• May become trapped in local minima • Less likely to find optimal policy • Data must be collected following current policy	• Policy learned may not be as safe • May not perform as well online

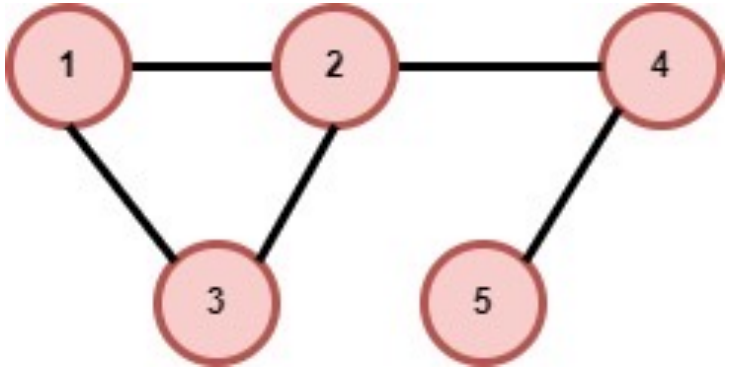
https://core-robotics.gatech.edu/2022/02/28/bootcamp-summer-2020-week-4-on-policy-vs-off-policy-reinforcement-learning/

Graph Neural Networks

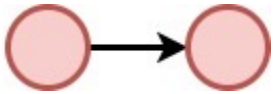
Material from https://distill.pub/2021/gnn-intro/

& a presentation by Sena Eşme

A Simple Graph



undirected



directed

- G = (V,E)
- V – Vertex
- E – Edge

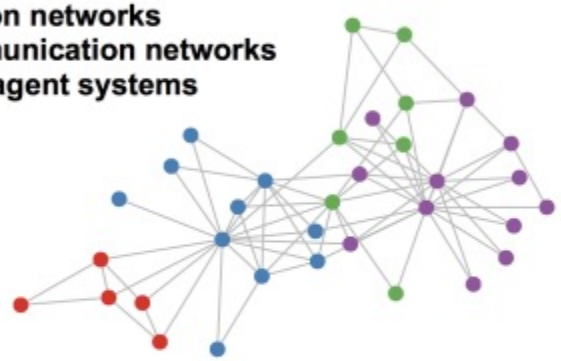
	V1	V2	V3	..
V1	0	1	1	
V2	1	0	1	
V3	1	1	0	
..				

Adjacency matrix
VxV

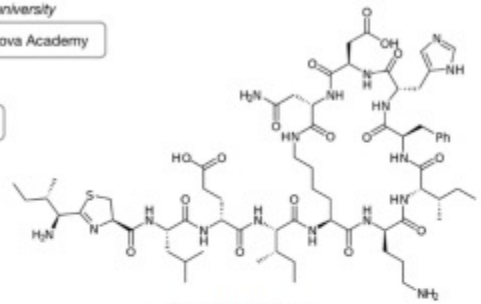
Material: https://distill.pub/2021/gnn-intro/

Graph Data

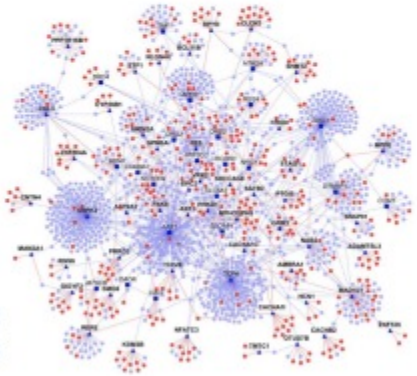
Social networks
Citation networks
Communication networks
Multi-agent systems



Knowledge graphs



Molecules



Protein interaction networks

Road maps

Material: https://distill.pub/2021/gnn-intro/

Images as Graphs

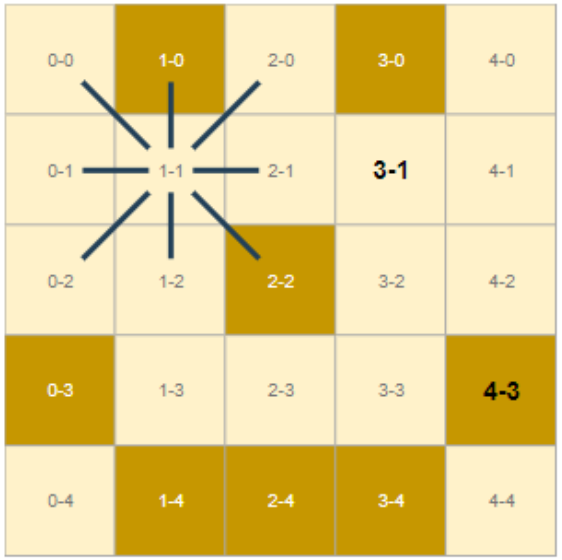
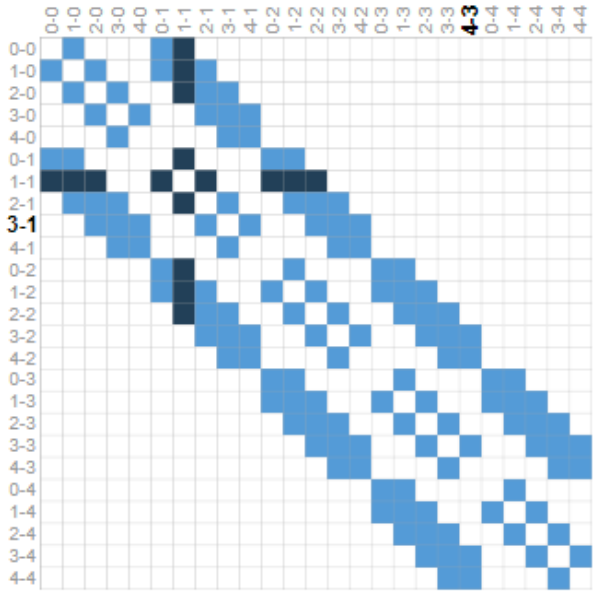
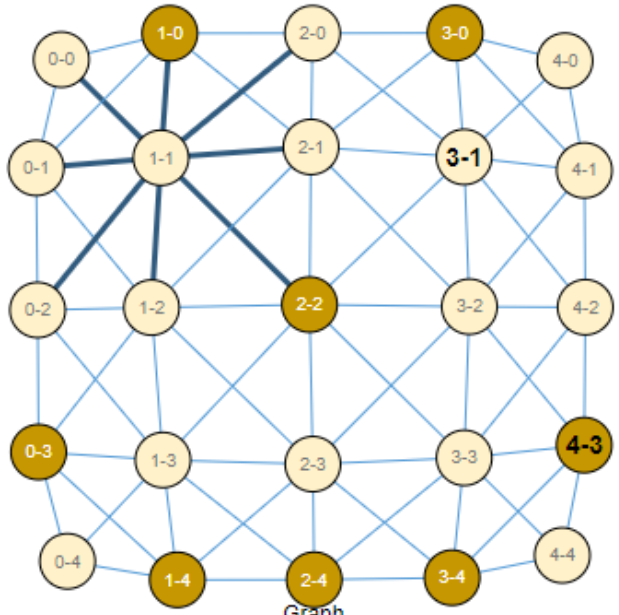


Image Pixels



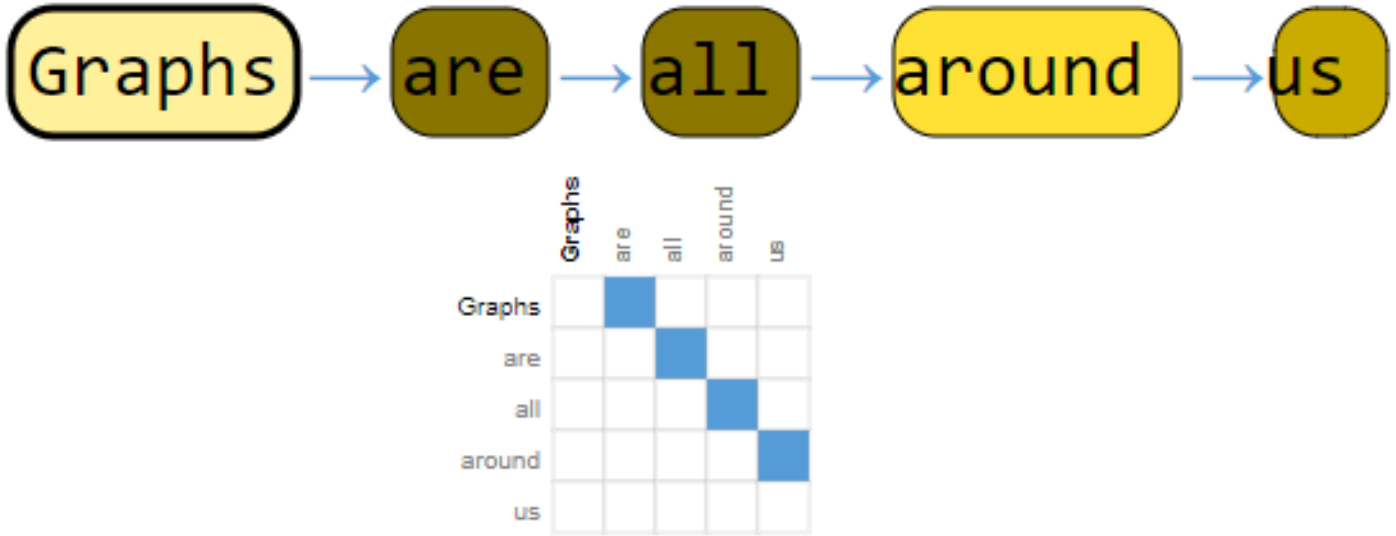
Adjacency Matrix



Graph

Material: https://distill.pub/2021/gnn-intro/

Text as Graphs



Material: https://distill.pub/2021/gnn-intro/

What types of problems have graph structured data?

Three general types of prediction tasks on graphs:

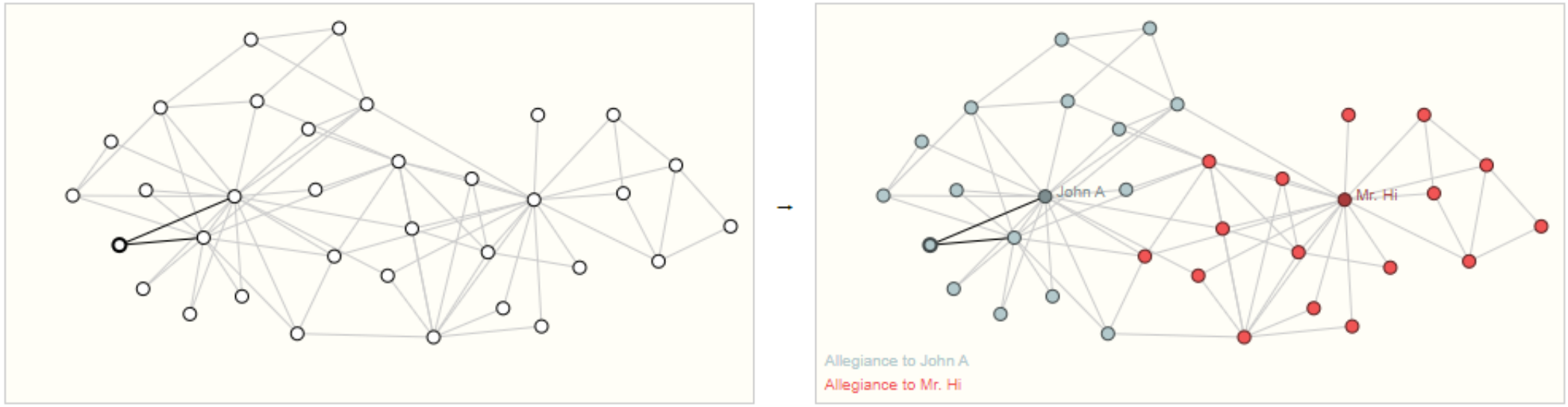
- Node-level
- Edge-level
- Graph-level

Material: https://distill.pub/2021/gnn-intro/

Node-level

- Predict attributes about nodes or classify them

Zach's Karate Club



Input: graph with unlabled nodes

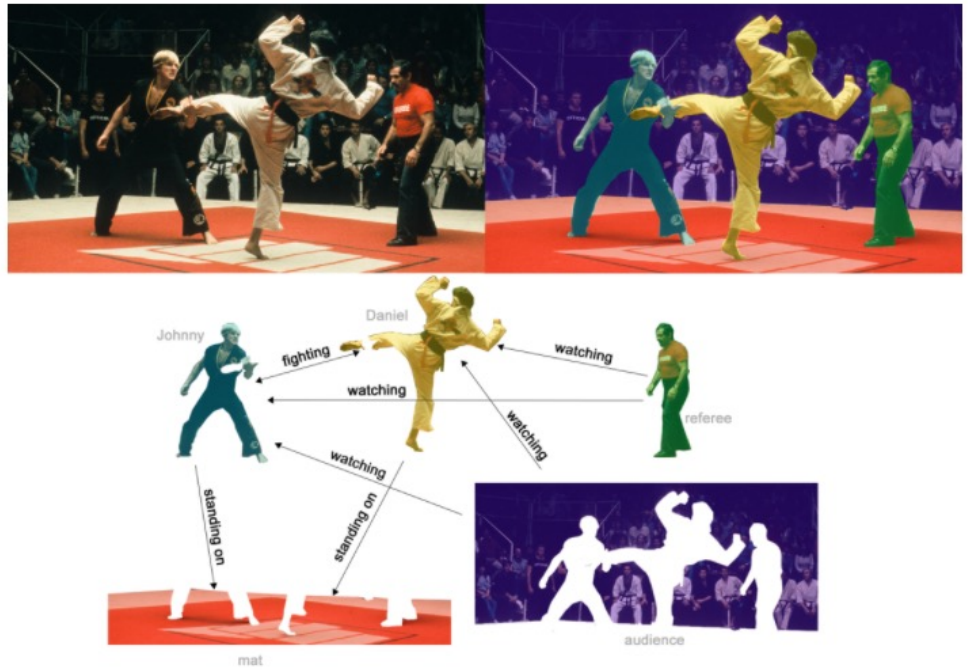
Output: graph node labels

Material: https://distill.pub/2021/gnn-intro/

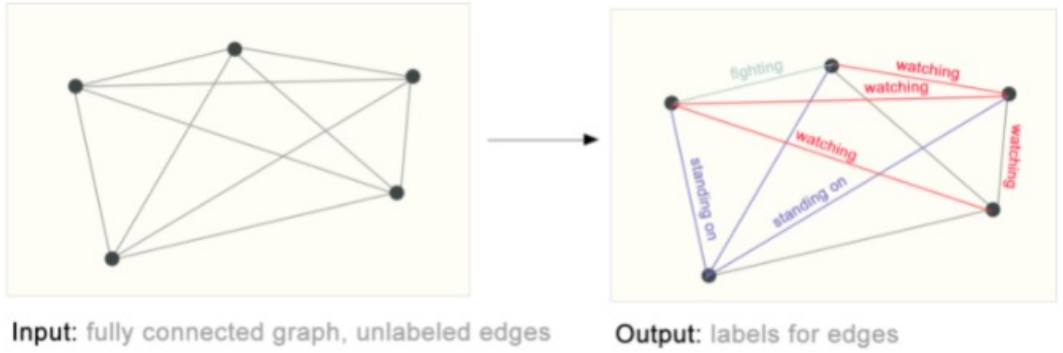
Edge-level

Predict whether there is a connection between two nodes

- Social networks to infer social interactions or to suggest possible friends to the users
- Predicting which customer will buy which product next
- Image scene understanding



In (b), above, the original image (a) has been segmented into five entities: each of the fighters, the referee, the audience and the mat. (C) shows the relationships between these entities.



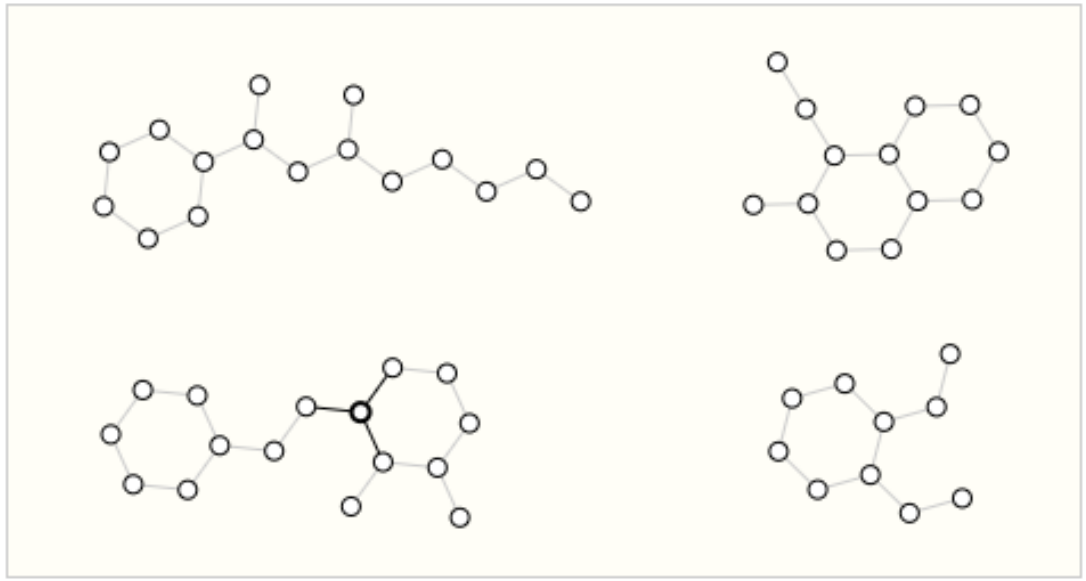
Input: fully connected graph, unlabeled edges

Output: labels for edges

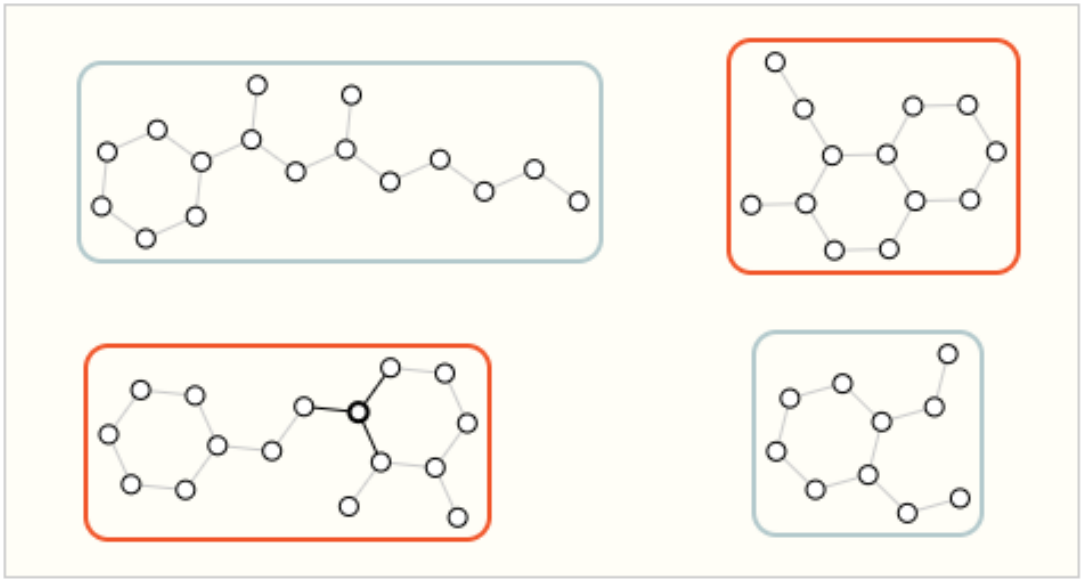
Material: https://distill.pub/2021/gnn-intro/

Graph-level

- Classifying entire graphs.
  - Predicting the molecule



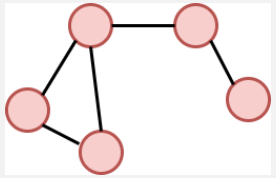
Input: graphs

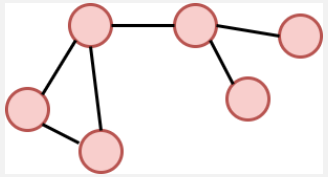


Output: labels for each graph, (e.g., "does the graph contain two rings?")

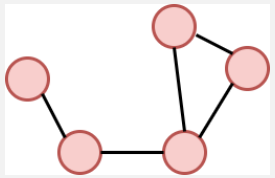
The challenges of using graphs in machine learning

1- Size and Shape

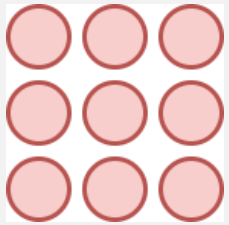


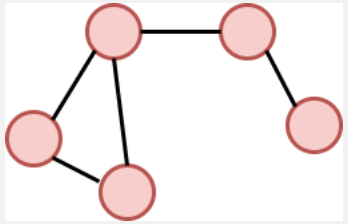


2- Isomorphism



3- Grid Structure





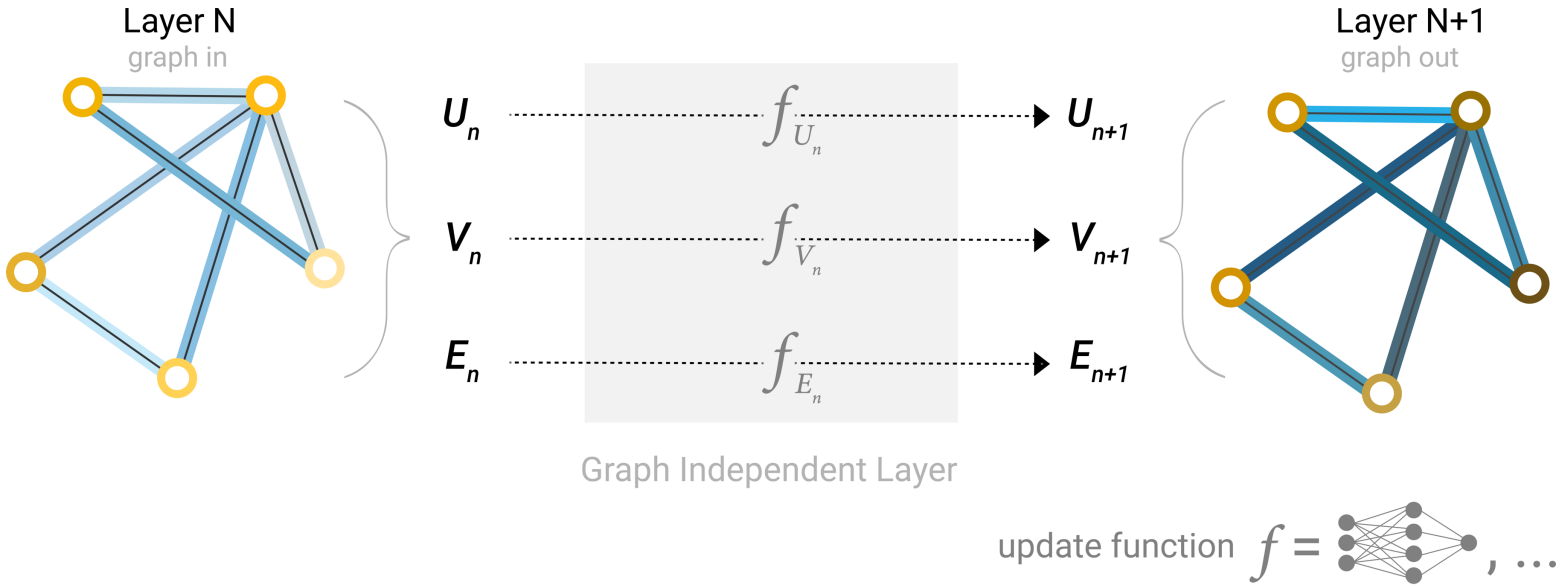
Non-Euclidean

Material: https://distill.pub/2021/gnn-intro/

Graph Neural Networks

- A GNN is an optimizable transformation on all attributes of the graph (nodes, edges, global-context) that preserves graph symmetries (permutation invariances).

The Simplest GNN



graph-in, graph-out
Seperate MLP on each component
Message passing neural network

Material: https://distill.pub/2021/gnn-intro/

GNN Predictions by Pooling Information

- Consider the task is to make binary classification on nodes,
- Graph contains node information
- For each node embedding do linear classifier.

BUT WHAT IF THERE IS AN INFO IN EDGES?

- Collect info from edges and give them to nodes -> pooling

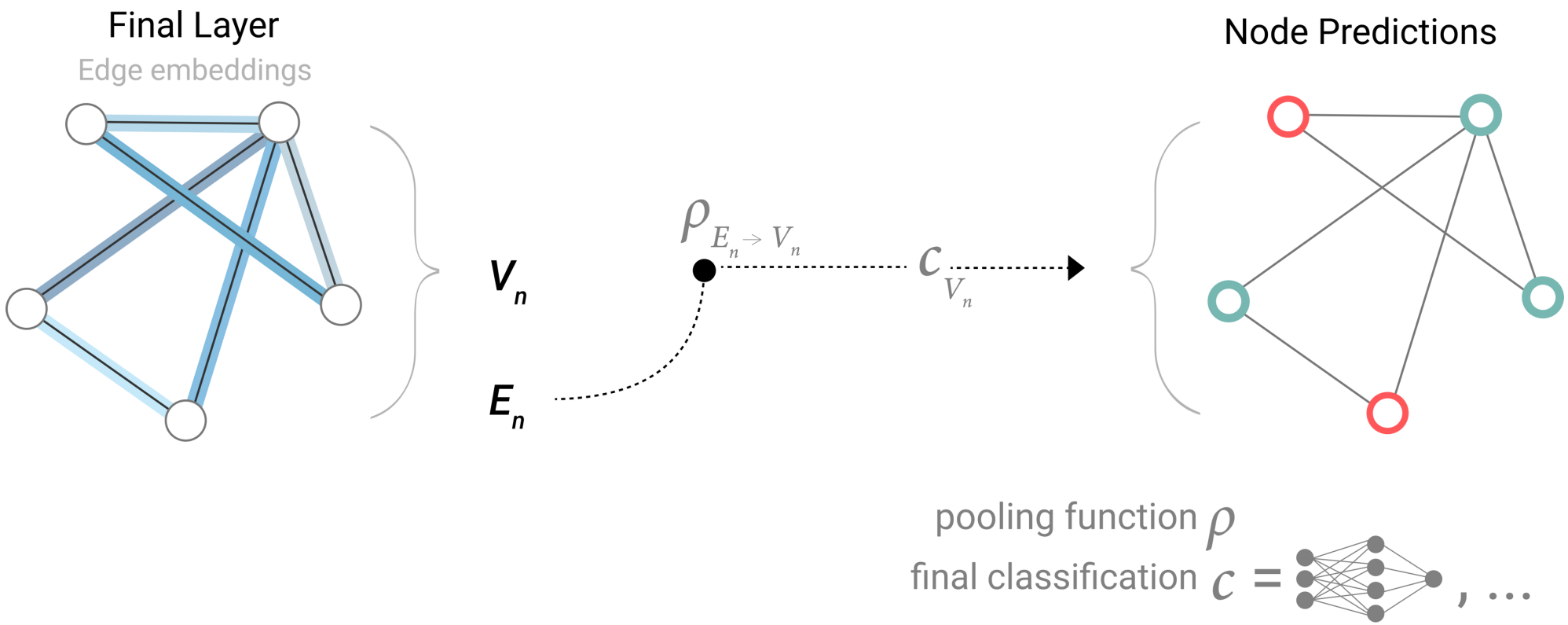
Pooling:

1- For each item to be pooled, gather each of their embeddings and concatenate them into a matrix.

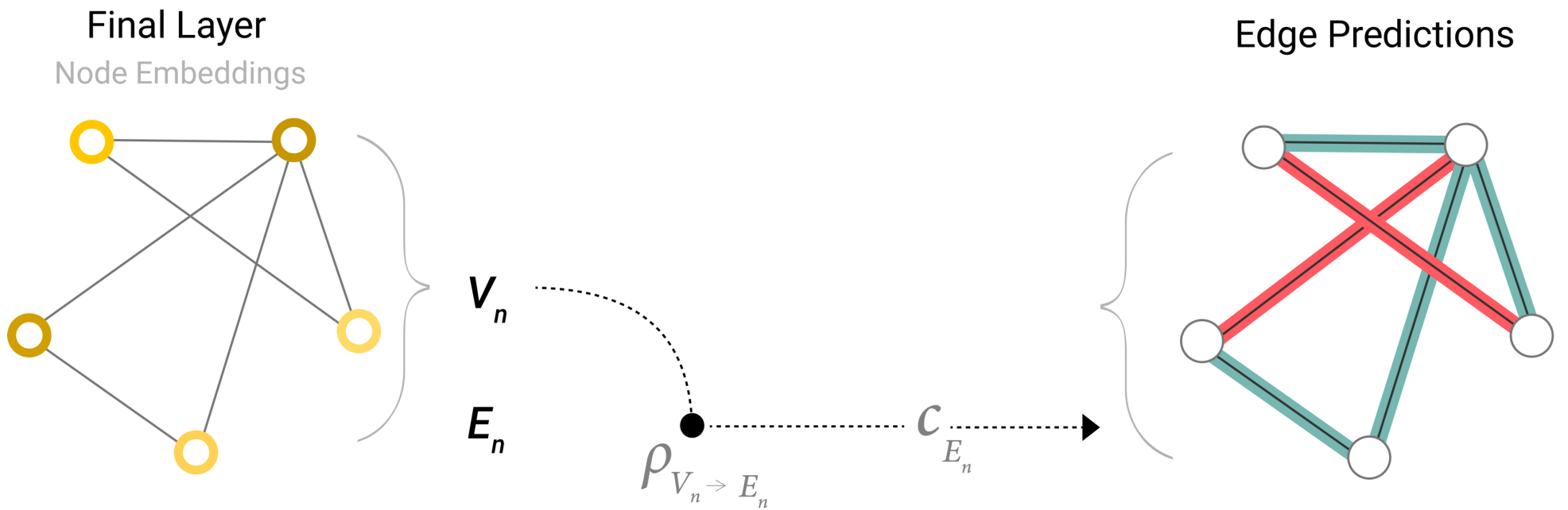
2- The gathered embeddings are then aggregated, via a sum operation.

$$p_{E_n \to V_n} \cdot$$

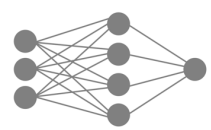
have edge-level features, predict binary node information



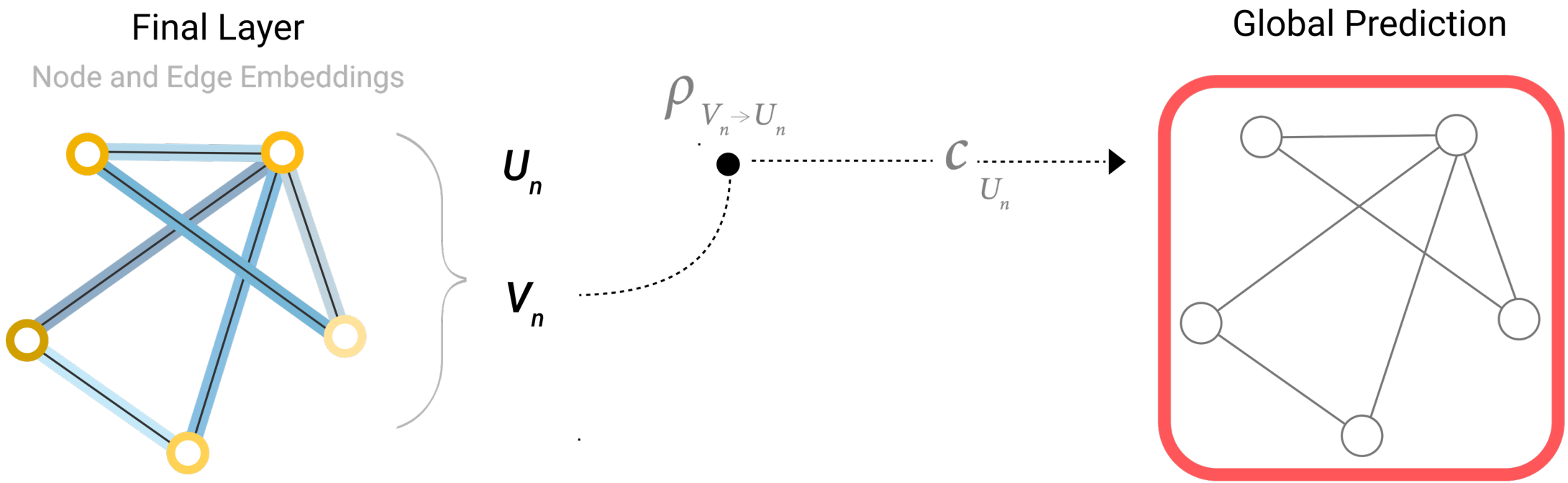
have node-level features, predict binary edge information



pooling function ρ

final classification c =  , ...

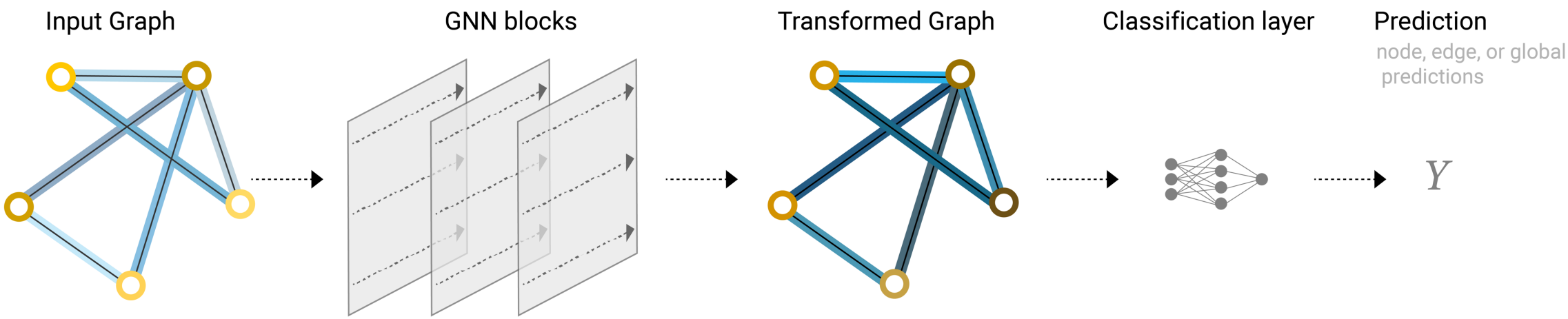
have node-level features, predict binary global information



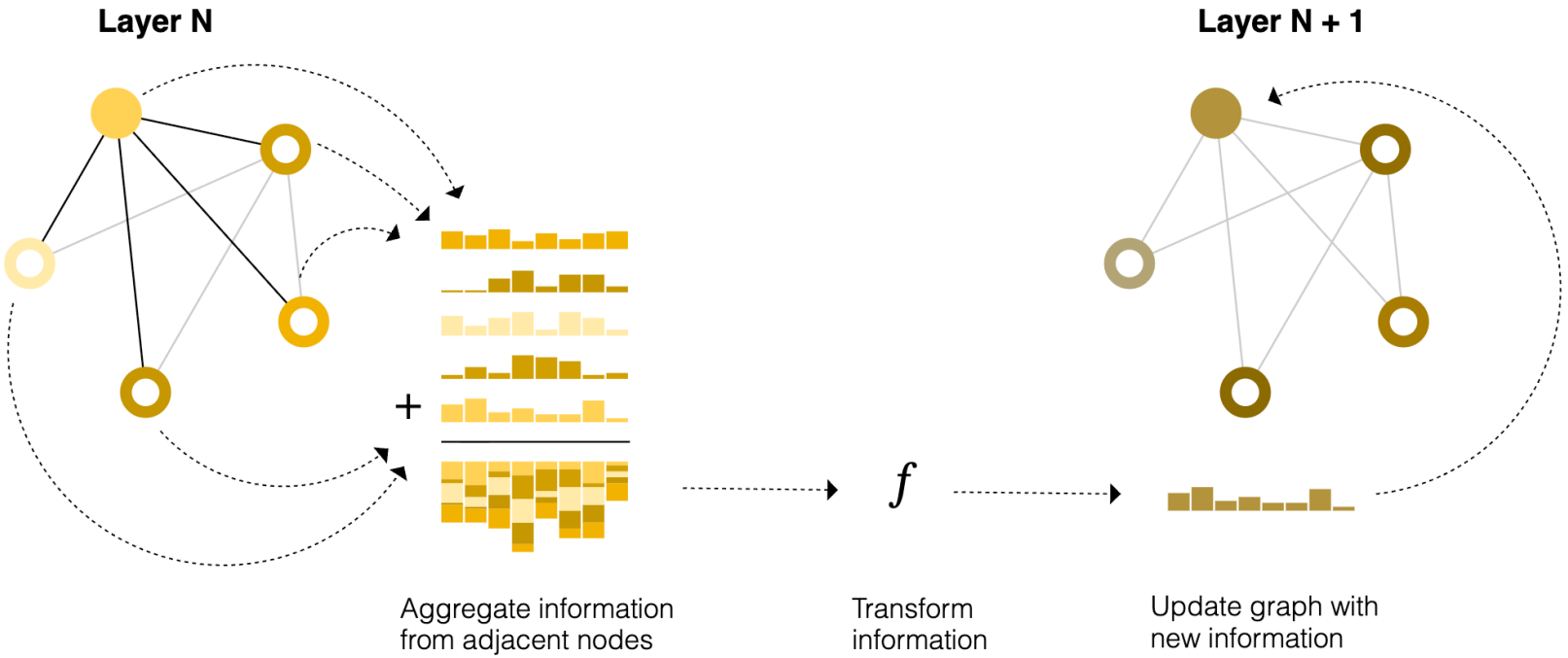
pooling function ρ

final classification c = , ...

Overall Structure of GNNs



Message Passing

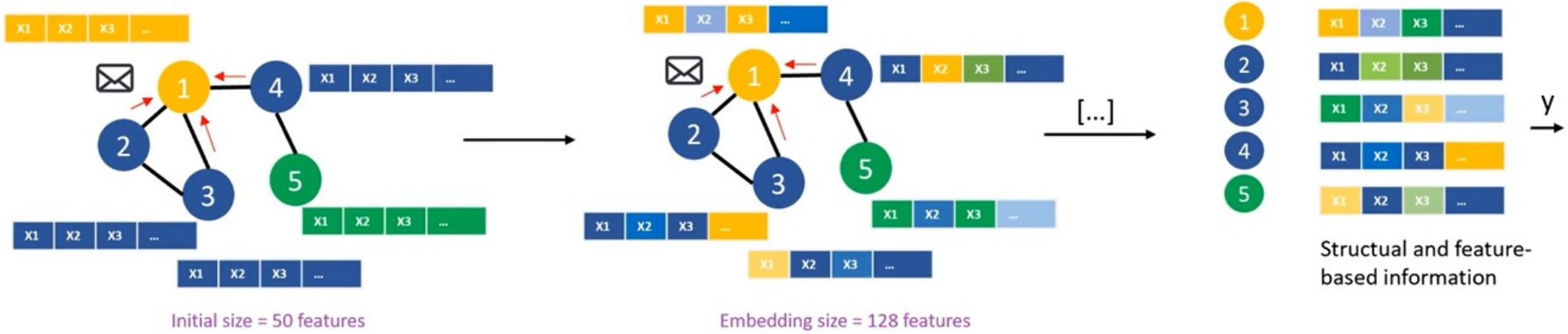


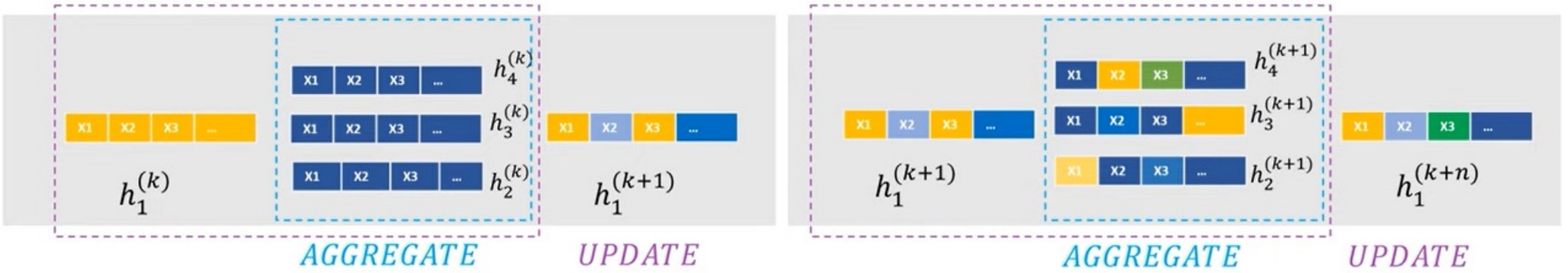
Message passing works in three steps:

1- For each node in the graph, gather all the neighboring node embeddings.
2- Aggregate all messages via an aggregate function (like sum).
3- All pooled messages are passed through an update function.

$$h_u^{(k+1)} = UPDATE^{(k)}\left(h_u^{(k)}, AGGREGATE^{(k)}(\{h_v^{(k)}, \forall\, v \in \mathcal{N}(u)\})\right)$$

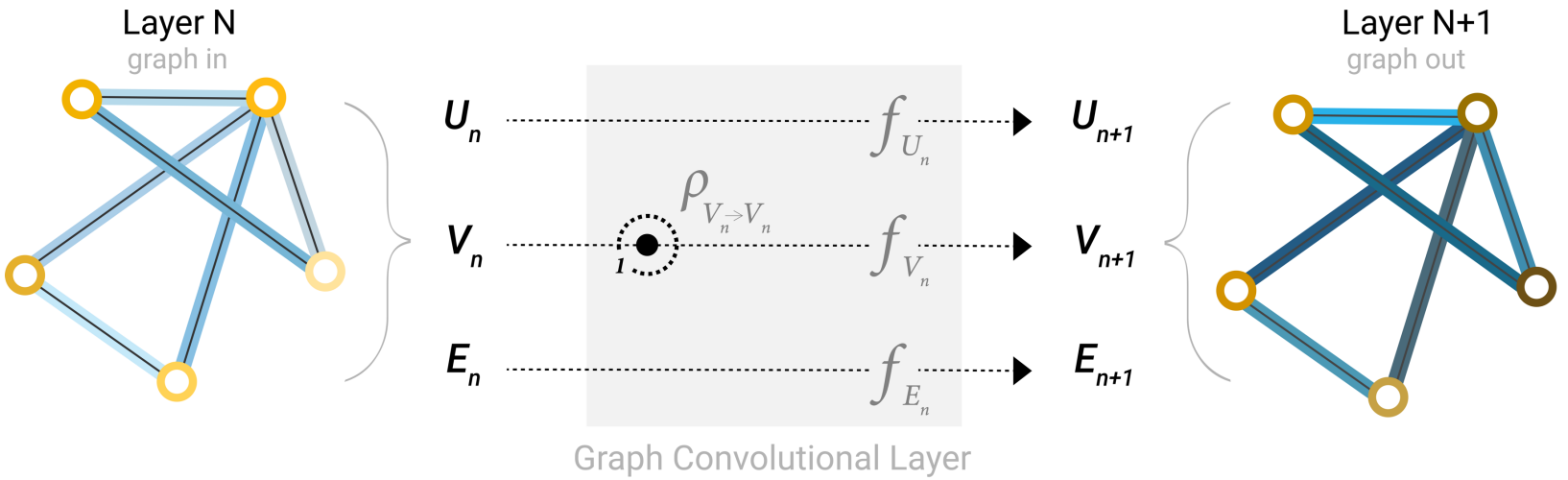
Message Passing





Learning Edge Representations

- Task is node prediction, but we only have edge information.
- Pooling is only at the final prediction step.
- Within GNN, use message passing.
- But the problem different size of node and edge information.
- Solution: Linear mapping from space of edges to space of nodes or concatenate them together before update.



update function $f =$, …

pooling function ρ

Adding Global Representation

- Nodes that are far away from each other may never be able to efficiently transfer information to each other.
- If all nodes be able to pass information to each other, computationally expensive
- Solution: Using global representation of a graph (U), master node or contex vector.