Building deep retrieval models in practical applications

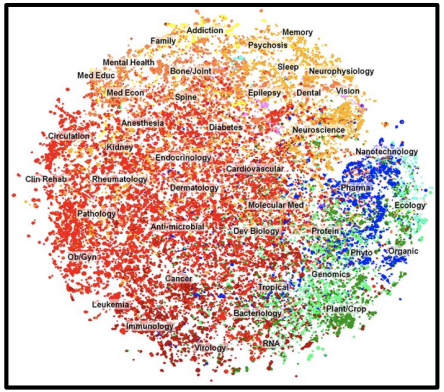
VinAI, Vietnam / *April 1st, 2022*

Khoa D. Doan [khoadoan.me]
Department of Computer Science, **Virginia Tech**
Cognitive Computing Lab, **Baidu Research, USA**

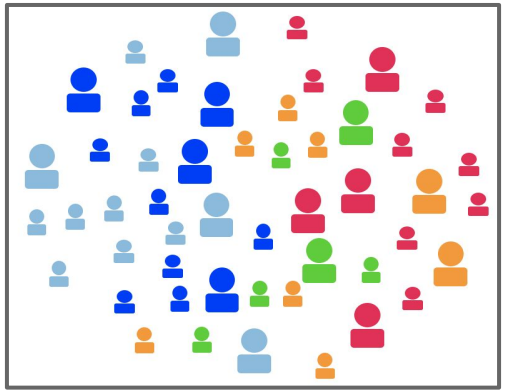
OUTLINE

- The retrieval problem
- Hashing Approaches
  - Single-loss quantization
  - Energy-based Hashing Networks
- Interpretable Retrieval
- Thoughts on Hashing
- Q & A!

Retrieving data is around us…

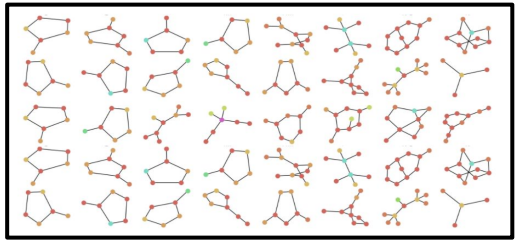


Document Retrieval
[Salakhutdinov et al. 2009]



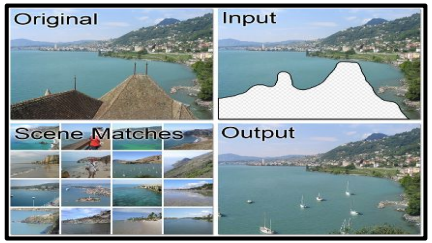
Similar Users Search
[Doan et al. 2019]

Image Retrieval
[Doan et al. 2022]



Graph Search
[Doan et al. 2021]

Fingerprint Matching



Near-duplicate Detection
[Henzinger et al. 2006]

Scene Completion

Retrieval & Similarity Search

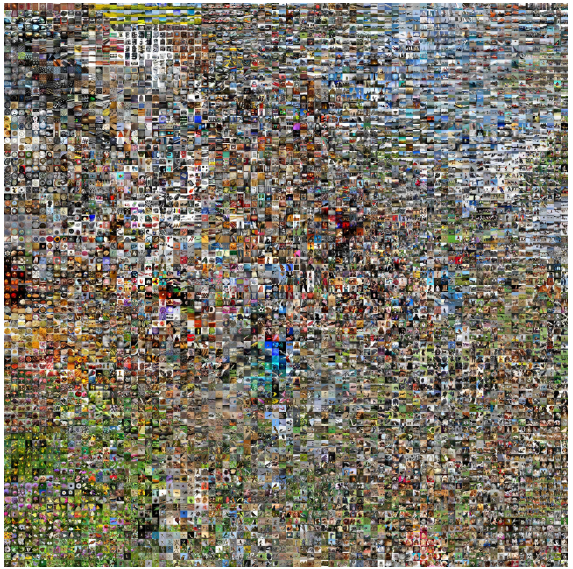
Problem: Given a dataset of N items $X = \{x_1, x_2, \ldots, x_N\}$ and a query q, we aim to find l items $R = \{x_1, x_2, \ldots, x_l\}$ such that, for a similarity function sim, we have:

$$sim(q, x_i) \geq sim(q, x_j)$$
$$\forall x_i \in R, \forall x_j \in X \backslash R$$

query

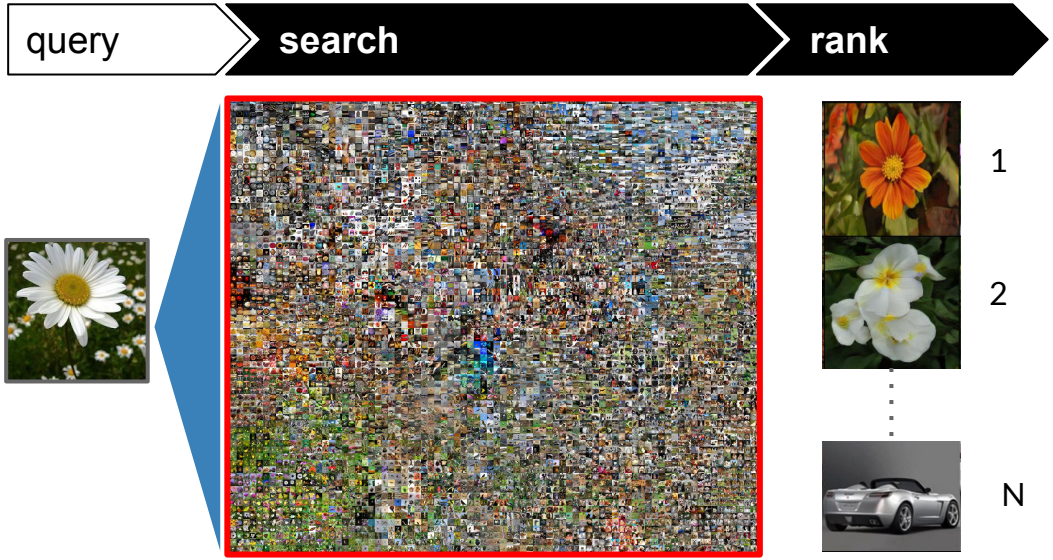
find similar images

search results



large image database

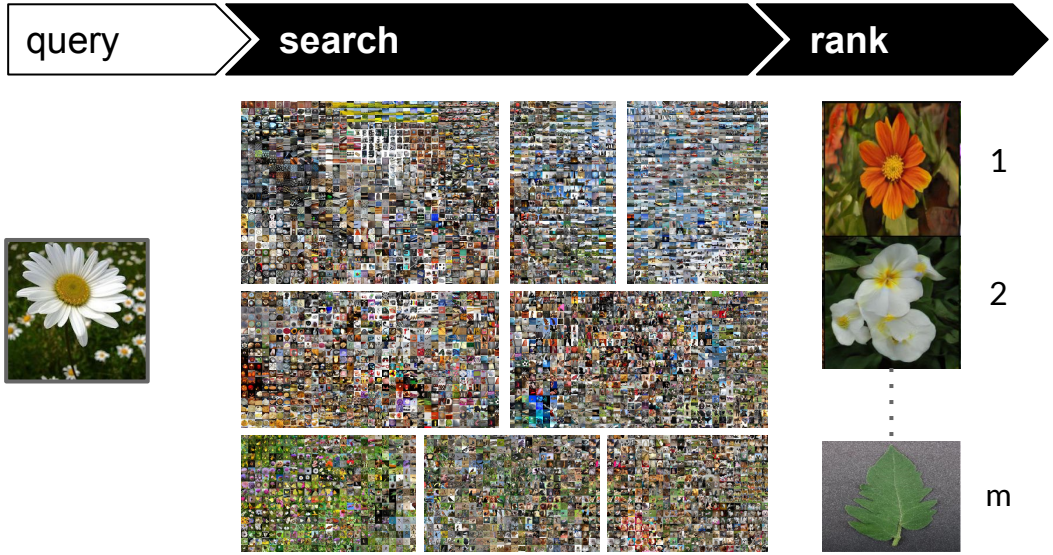
Linear Search



Exhaustive search

- infeasible in large database of millions or billions of items.
- wasteful of computation
  - only a small subset is relevant
  - real-time ranking is impossible

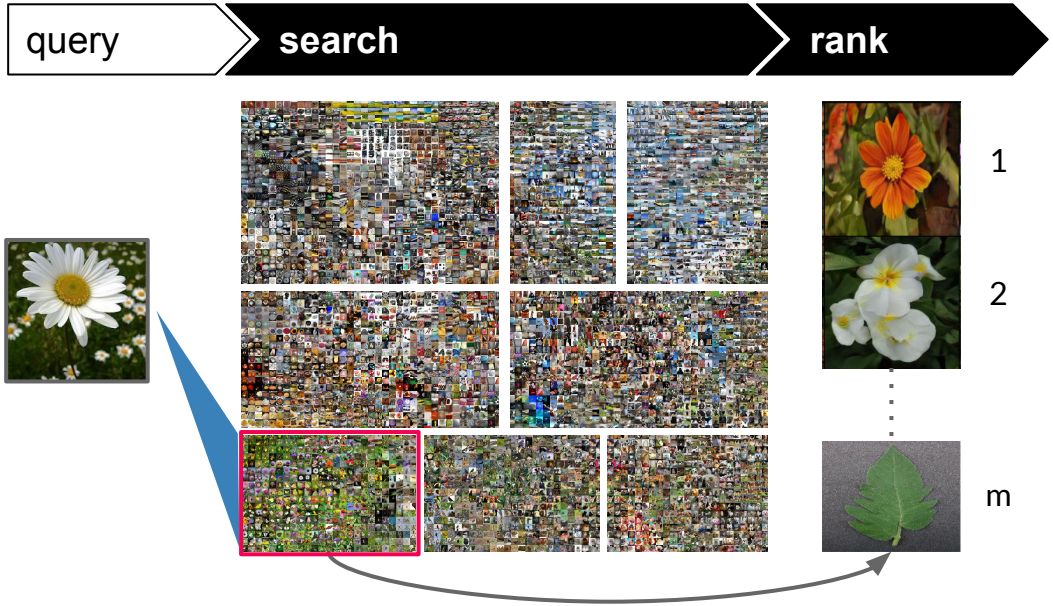
Approximate nearest neighbor



Approximate Search

- ANN search builds an index structure

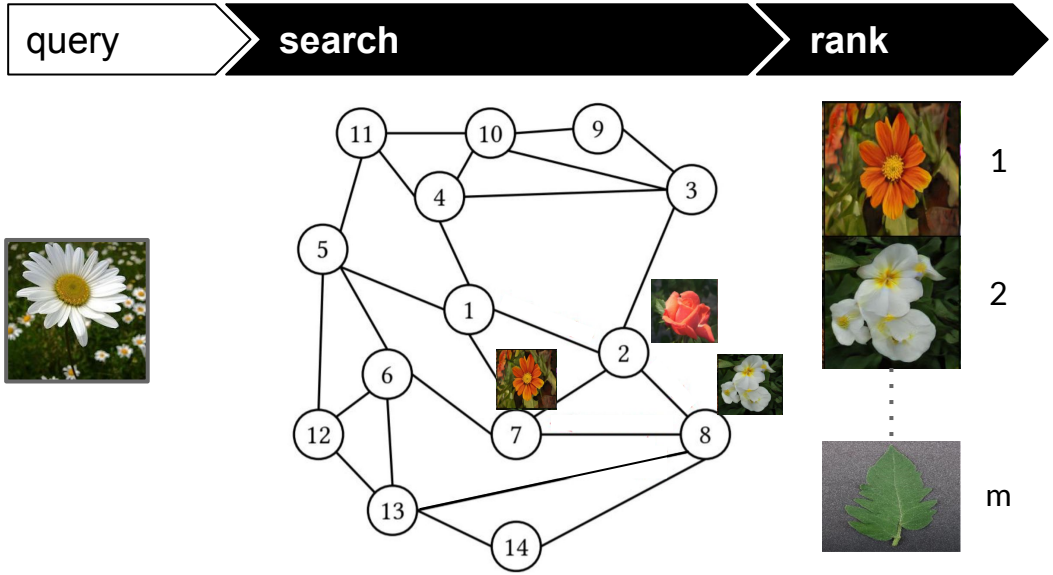
Approximate nearest neighbor



Approximate Search

- ANN search builds an index structure
  - limits the search to a subset of candidate items (**sub-linear**)
- What if the index is not perfectly constructed?

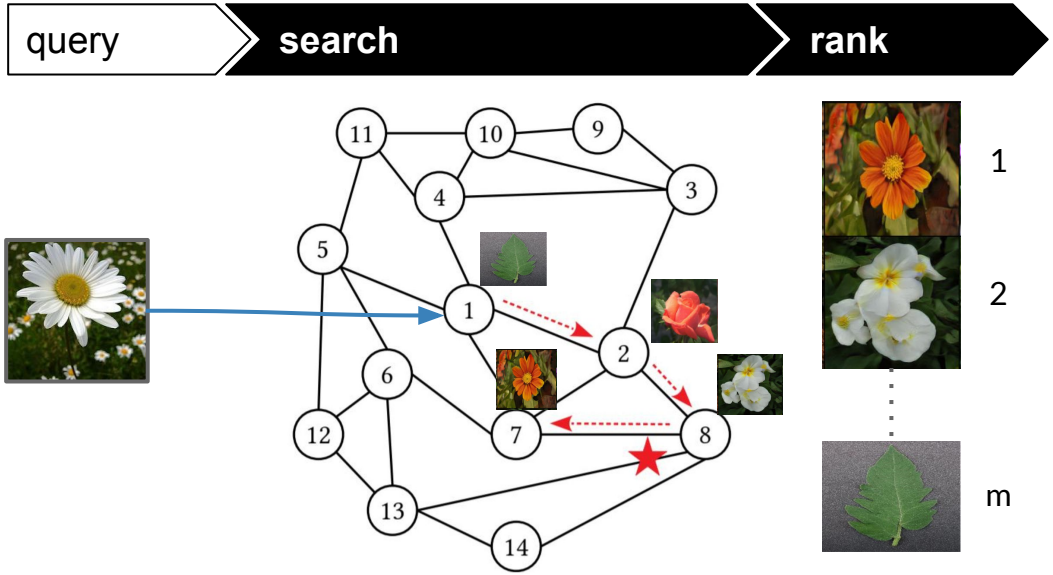
Approximate nearest neighbor



Approximate Search (Graph)

- Construct a proximity graph of items

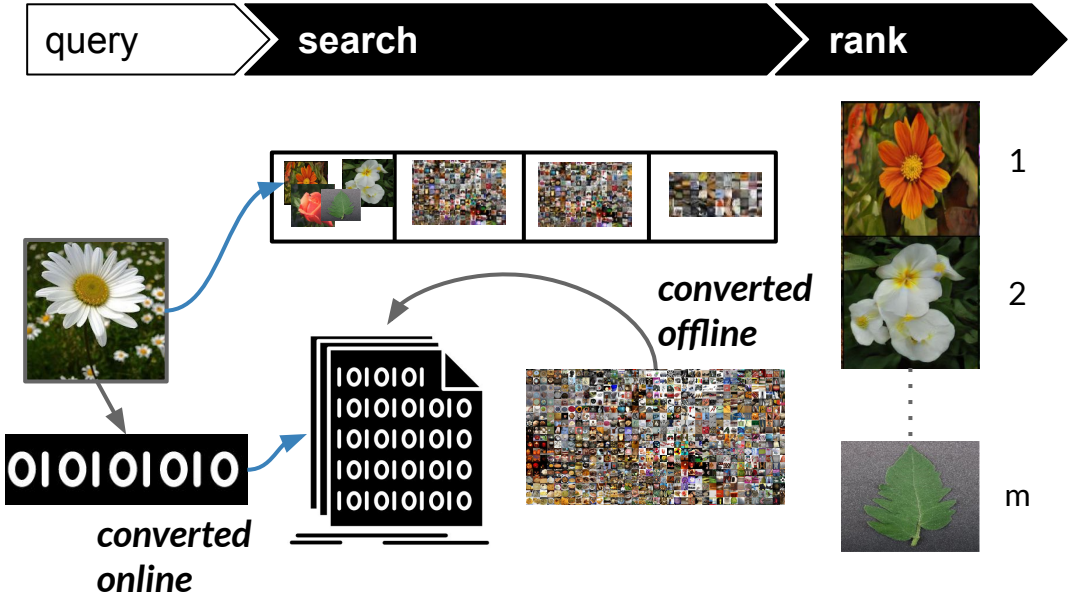
Approximate nearest neighbor



Approximate Search (Graph)

- Construct a proximity graph of items
- Search becomes traversing the graph
  - *Compute some distance at each vertex*

Approximate nearest neighbor

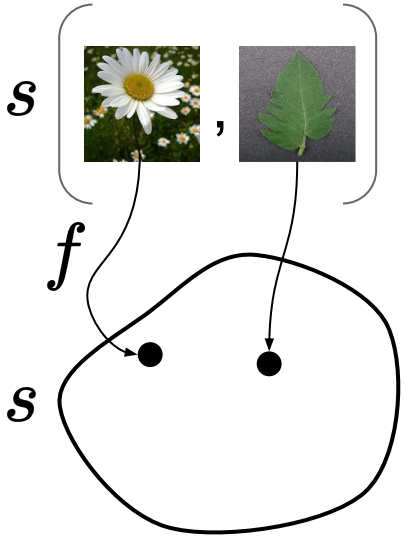


Approximate Search (Hashing)

- Transforms images into binary vectors
- Search via table look-up
- Linear Search in Discrete space:
  - Memory efficient: 4MB for 1M items
  - Compute efficient: 2 instructions per distance computation
  - Can be parallelized in distributed systems
  - **Avoid original similarity computation, thus more computationally efficient than graph-based ANN**

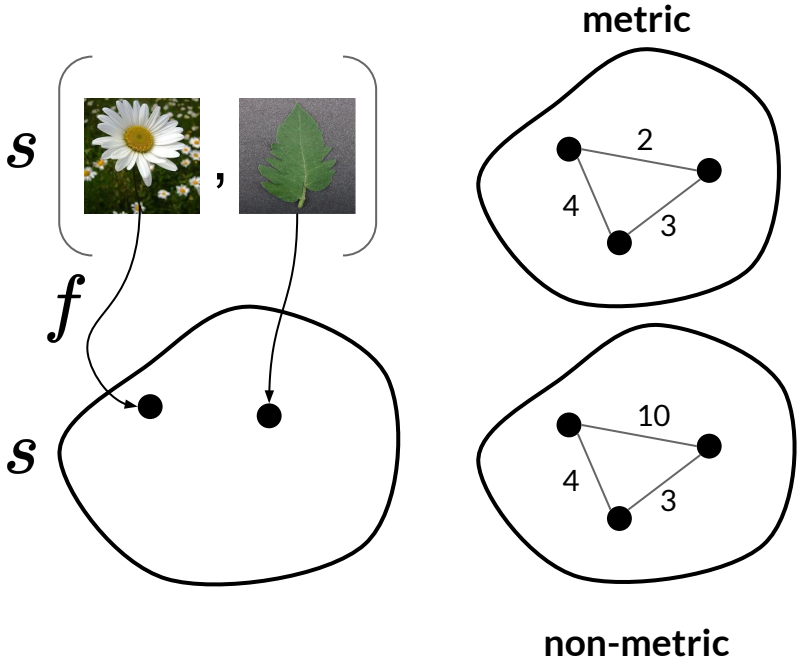
How to define the similarity function?

- Explicit representations of items
  - Input space
  - Representation space



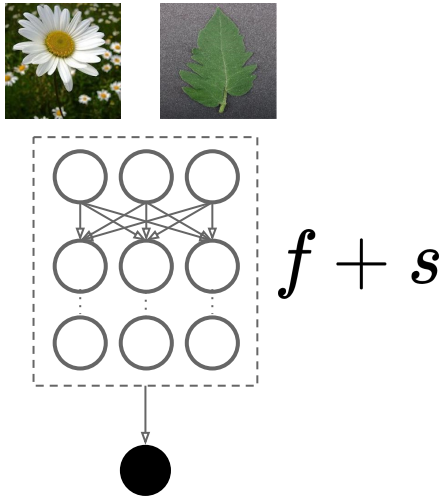
How to define the similarity function?

- Explicit representations of items
  - Input space
  - Representation space
  - Popular distance measures such as
    - Metric: Euclidean, Cosine…
    - Non-metric: Inner Product (MIPS Problems) [Shrivastava et al. 2014]



How to define the similarity function?

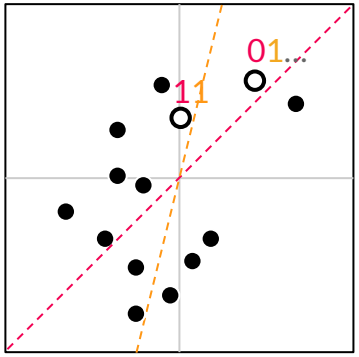
- Explicit representations of items
  - Input space
  - Representation space
  - Popular distance measures such as
    - Metric: Euclidean, Cosine…
    - Non-metric: Inner Product (MIPS Problems) [Shrivastava et al. 2014]
- Black-box distance measure: general non-linear, non-metric
  - Neural Network [Tan et al. 2020]



Quick Recap

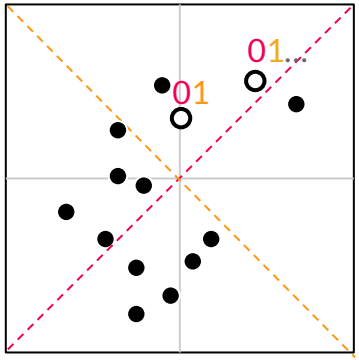
- Quickly discussed
  - Retrieval applications
  - Linear search vs. Approximate Search
  - How is similarity defined
- In this talk:
  - Hashing approaches that learn the hash function.
    - When the similarity function is not defined
  - Retrieval approach that explains its decision.
  - Applications of Hashing in other tasks.

Hashing Approaches



Data Independent

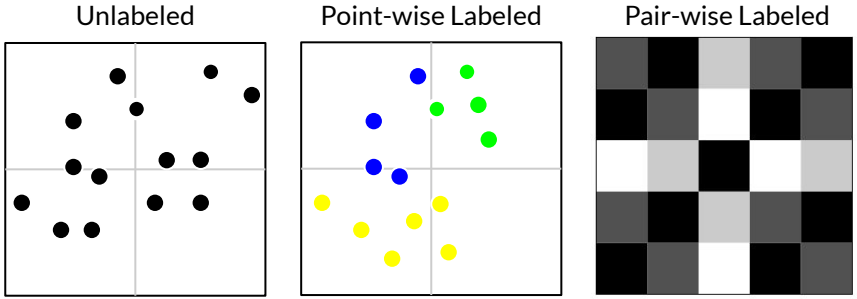
- Locality Sensitive Hashing:
  MinHash [Li et al. 2011]
  Sign Random Projection
  [Li et al. 2014]



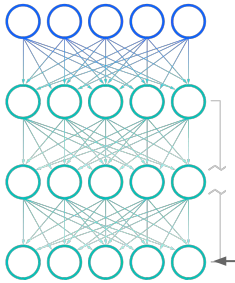
Data Dependent

- Spectral Hashing
  [Weiss et al. 2009],
- Iterative Quantization
  Gong et al. 2012]

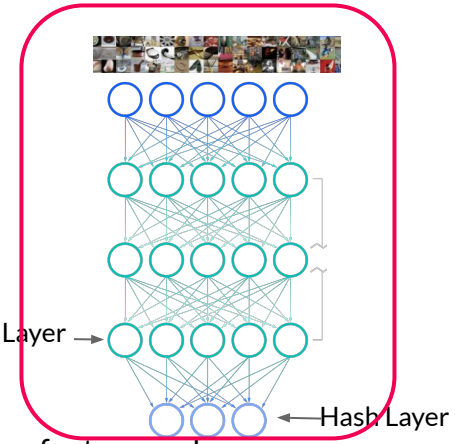
Unsupervised or Supervised



Shallow or Deep



Learn similarity
from **fixed** features



Learn feature and
similarity end to end

Hash-function learning

quantized during inference (i.e., > 0.5)

- Learn a hash function

$$F : \mathcal{R}^n \longrightarrow \{0, 1\}^m \rightarrow F : \mathcal{R}^n \longrightarrow [0, 1]^m$$

discrete optimization — continuous relaxation

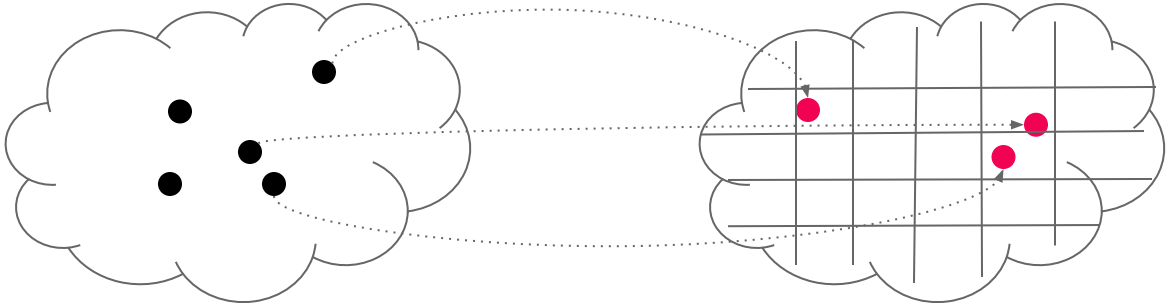
- Overall objective function of hashing methods

$$\arg\min_f E_{x \sim D_x} L(x, f(x)) + E_{x \sim D_x} \sum_k \lambda_i \times H_k(f(x))$$

locality-preserving loss — hashing regularizer

- **Locality loss** preserves the semantics of sim in discrete space
- **Heuristic regularizers** minimize the gaps between continuous and discrete optimizations.

Locality Preservation



Several definitions (depends on models and problems)

$$x_i \in \mathcal{N}_d(x) \implies F(x_i) \in \mathcal{N}_{\tilde{d}}(F(x))$$

Typically cosine/euclidean

Usually hamming distance

- **Example:** Given (x, x^-, x^+)

Dissimilar point

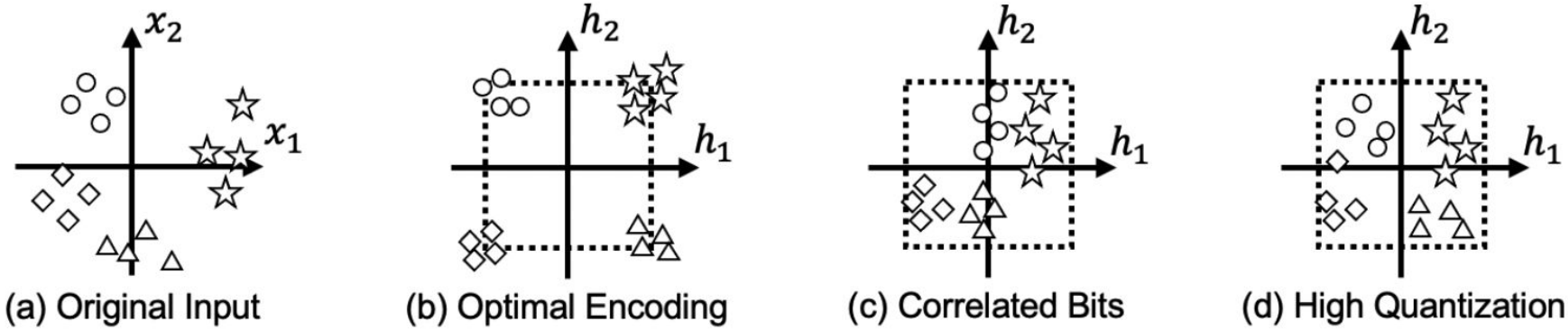
curren point

similar point

- Similar/Dissimilar: same class/different class
- Similar/Dissimilar: nearest neighbor/distant neighbor

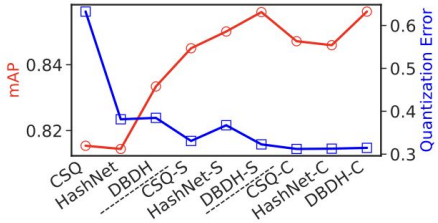
$$\min_\theta \sum_x \max(0, 1 + |F(x) - F(x^+)|_2 - |F(x) - f(x^-)|_2)$$

Quantization Regularization helps efficiency

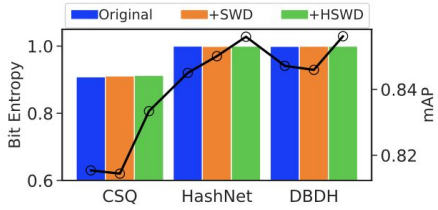


(a) Original Input (b) Optimal Encoding (c) Correlated Bits (d) High Quantization

Figure 1. A visual illustration of the optimal function (Figure (b)) and learned hash function with poor the code balance (Figure (c)) and quantization constraint (Figure (d)). Figure (a) shows the original data clusters from four classes.



(a) Quantization Error



(b) Bit Entropy

Better Quantization leads to Better Performance

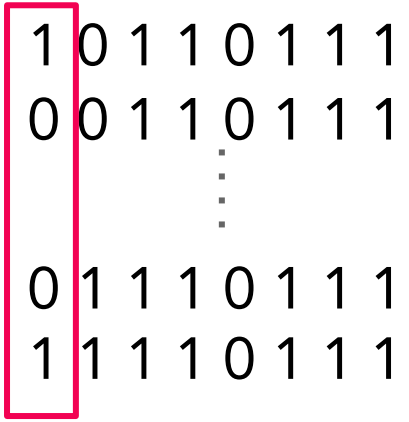
[Doan et al. 2022]

Achieving optimal quantization

- Bit Balance

$$\min_{\theta} \sum_{k} \bar{b}_k \log(\bar{b}_k) + (1 - \bar{b}_k) \log(1 - \bar{b}_k)$$

- Bit Uncorrelation
- Low-quantization Error



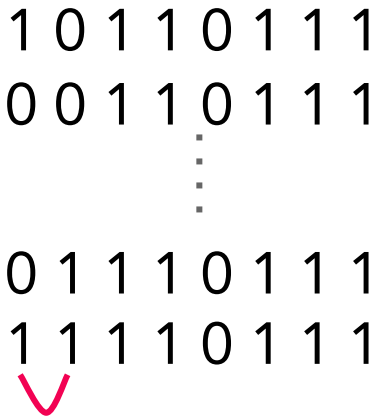
50% being 0 or 1

Achieving optimal quantization

- Bit Balance
- Bit Uncorrelation

$$\min_{\theta} |W^T W - I|_2$$

- Low-quantization Error



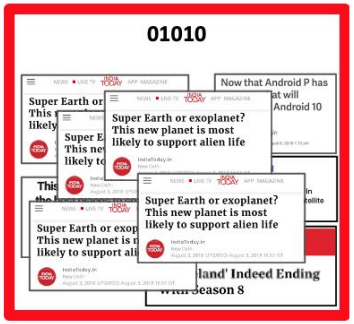
bits are not dependent

(i.e. encode different information)

Achieving optimal quantization

- Bit Balance
- Bit Uncorrelation
- Low-quantization Error

Code Balance



Bad: more work

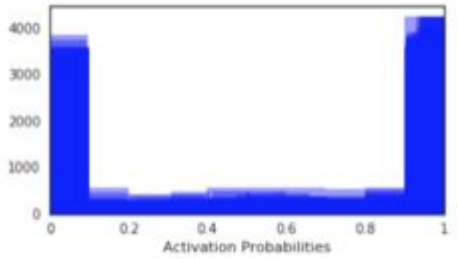
Good: when needing more examples, increase the hamming ball

Achieving optimal quantization

- Bit Balance
- Bit Uncorrelation
- Low-quantization Error

$$\min_{\theta} \sum_{i} \sum_{k} -b \log(b) - (1-b) \log(1-b)$$

0.9	0.1	…	0.1
0.1	0.2	…	0.3
		⋮	
0.1	0.3	…	0.1
0.2	0.1	…	0.1



Single-shot Quantization

Previous approaches: constraint several quantization losses on the learned distribution.

$$\arg\min_f E_{x\sim D_x} \sum_k \lambda_i \times H_k(f(x))$$

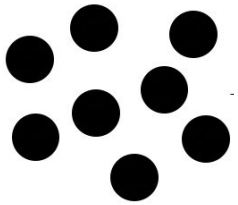
Advantages: easier optimization
Disadvantages: more hyperparameter tuning

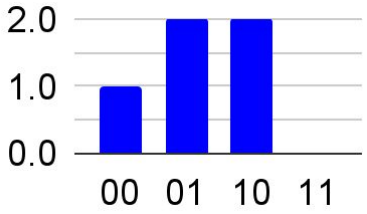
Our approach: directly minimizes a single divergence loss of 2 distributions.

$$\arg\min_f d(q(b)||\, q^\star(b))$$

Advantages: single-shot optimization
Disadvantages: can be challenging to optimize

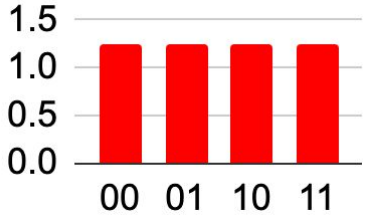
Task: learn 2-bit hash function. Optimal discrete distribution is one with equal-arrangement of the data into the binary codes





learned distribution q

$q^\star : b_i \sim \text{bernoulli}(0.5)$



optimal distribution $q^\star$
(with maximum entropy)

Single-shot Quantization

Previous approaches: constraint several quantization losses on the learned distribution.

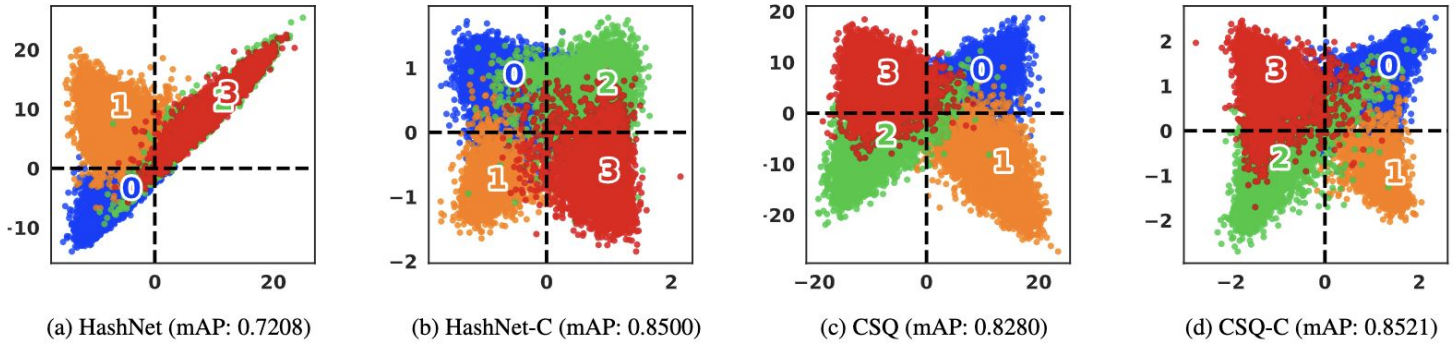
$$\arg\min_{f} E_{x\sim D_x} \sum_k \lambda_i \times H_k(f(x))$$

Advantages: easier optimization
Disadvantages: more hyperparameter tuning

Our approach: directly minimizes a single divergence loss of 2 distributions.

$$\arg\min_{f} d(q(b)||\, q^\star(b))$$

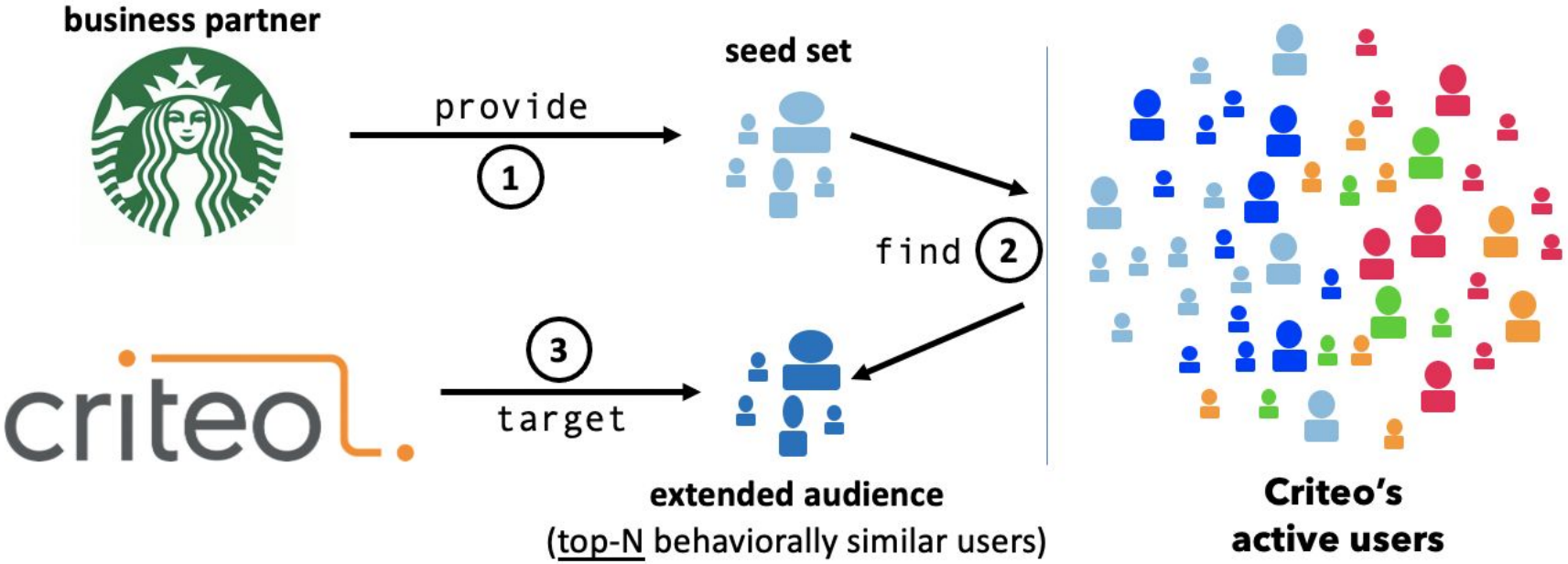
Advantages: single-shot optimization
Disadvantages: can be challenging to optimize



(a) HashNet (mAP: 0.7208) (b) HashNet-C (mAP: 0.8500) (c) CSQ (mAP: 0.8280) (d) CSQ-C (mAP: 0.8521)

Learn 2-bit hash function on CIFAR10's data from 4 classes

Application in Look-alike Modeling



[Doan et al. 2019]

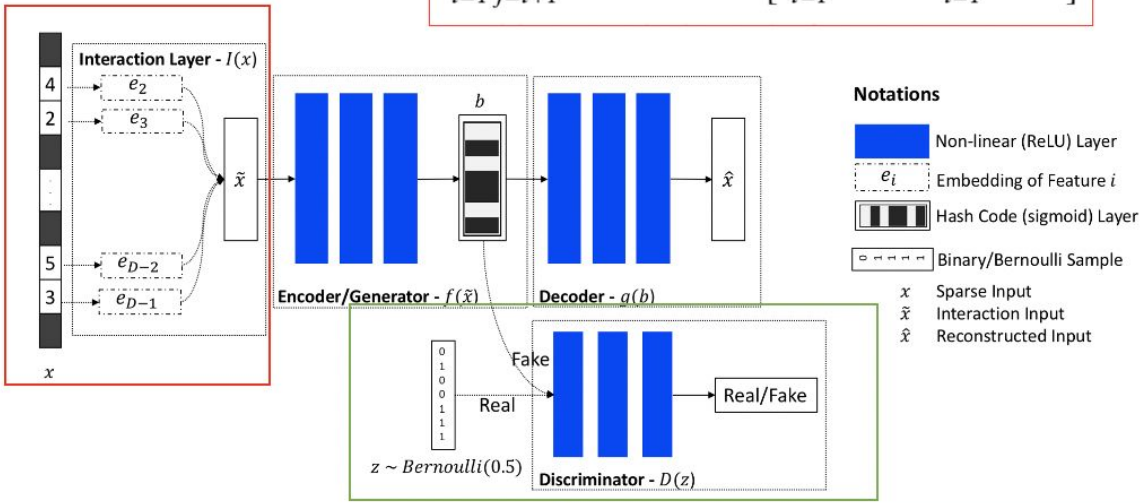
How to find look-alikes?

- Learn a classifier to determine if a user belongs to seed set
  - **Advantage**: good performance (with large seed sets)
  - **Disadvantage:**
    - time-consuming training (a model for every seed set)
    - severe overfitting (with less data)
- Learn an unsupervised retrieval model & find similar users
  - **Advantage**: one model for all seed sets
  - **Disadvantage:**
    - Completely unsupervised (may have poor performance)
    - We argue: poor representation learning + ineffective retrieval

Adversarial Factorization Autoencoder

learn sparse, high-dimensional data with high-order interaction

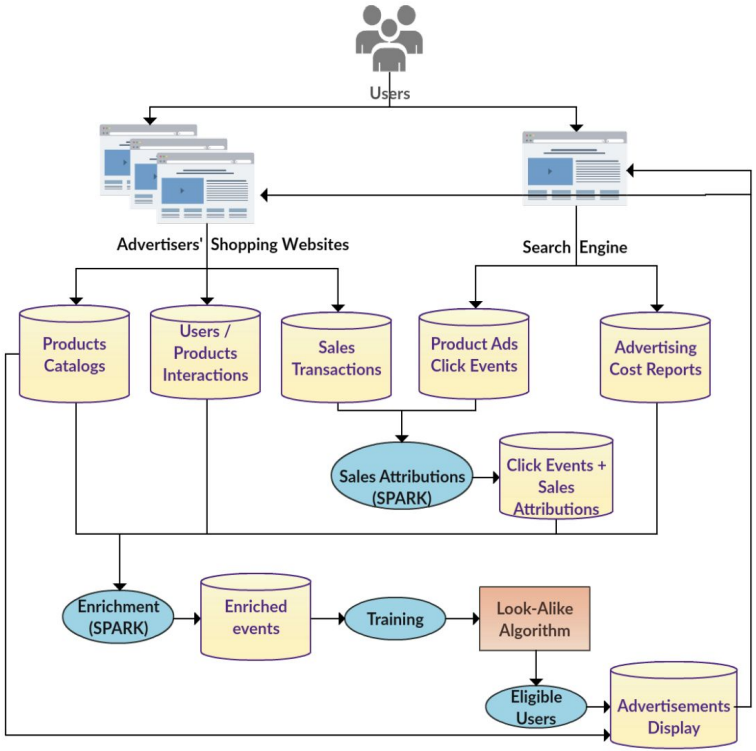
$$\sum_{i=1}^{n}\sum_{j=i+1}^{n} x_i e_i \odot x_j e_j = \frac{1}{2}\left[(\sum_{i=1}^{n} x_i e_i)^2 - \sum_{i=1}^{n}(x_i e_i)^2\right]$$



implicitly learns an optimal hash-function

$$\min_{\Theta_E}\max_{\Theta_D} L(E,D) = \min_{\Theta_E}\max_{\Theta_D} E_{z\sim Bernoulli_z(p)}[\log D(z)] + E_{x\sim D_x}[\log(1 - D(f(I(x))))]$$

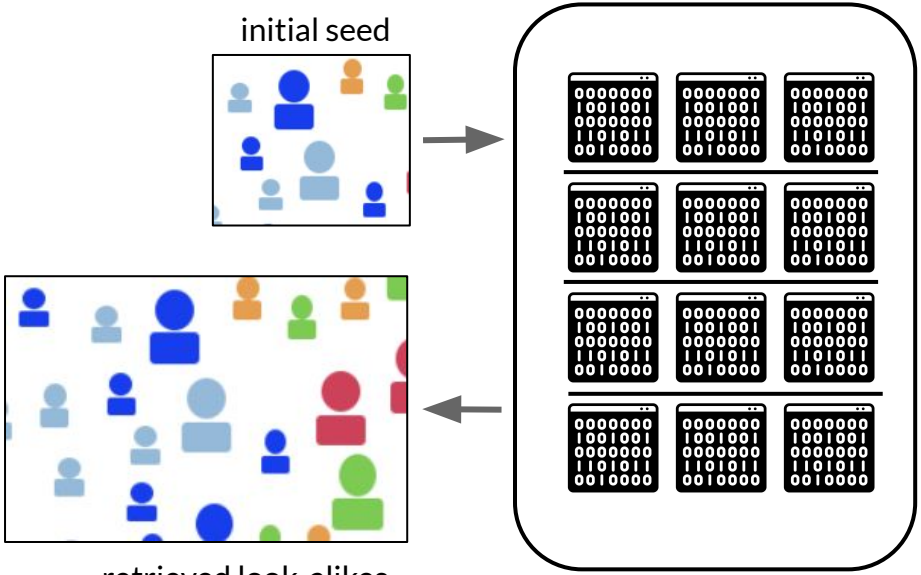
Deployed Retrieval-based Look-alike System



Detailed Processing Pipeline

Audience-extension System

initial seed



retrieved look-alikes

HOW DOES PERFORMANCE IMPROVE?

Recall w.r.t Ground-truth Look-alike users.

Dataset	Existing Hashing	Density Estimation	Classification	OURS
Apparel-1	0.466	0.522	0.604	**0.853**
Electronics-1	0.434	0.227	0.657	**0.843**
Home/Garden	0.457	0.541	0.631	**0.758**
Electronics-2	0.022	0.000	0.044	**0.652**

1. Achieves best performance when labeled data is scarce (**top 4**).

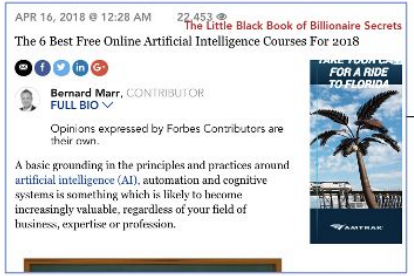
HOW DOES PERFORMANCE IMPROVE?

Recall w.r.t Ground-truth Look-alike users.

Dataset	Existing Hashing	Density Estimation	Classification	OURS
Apparel-1	0.466	0.522	0.604	**0.853**
Electronics-1	0.434	0.227	0.657	**0.843**
Home/Garden	0.457	0.541	0.631	**0.758**
Electronics-2	0.022	0.000	0.044	**0.652**
Animal/Pet	0.548	0.757	**0.941**	0.900
Apparel-2	0.638	0.757	**0.776**	0.766

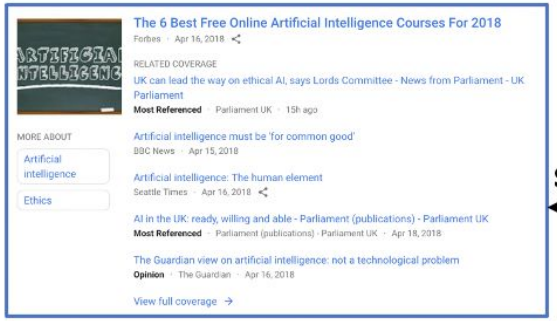
1. Achieves best performance when labeled data is scarce (**top 4**).
2. Achieves second-best when there are more seed data (**bottom 2**).

Application in Text Retrieval



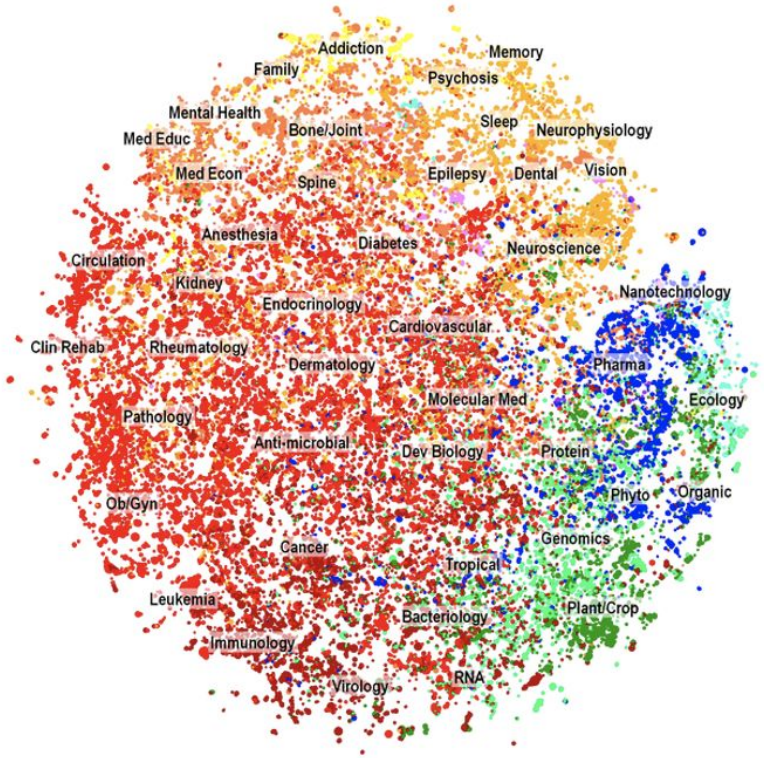
1

find other coverages



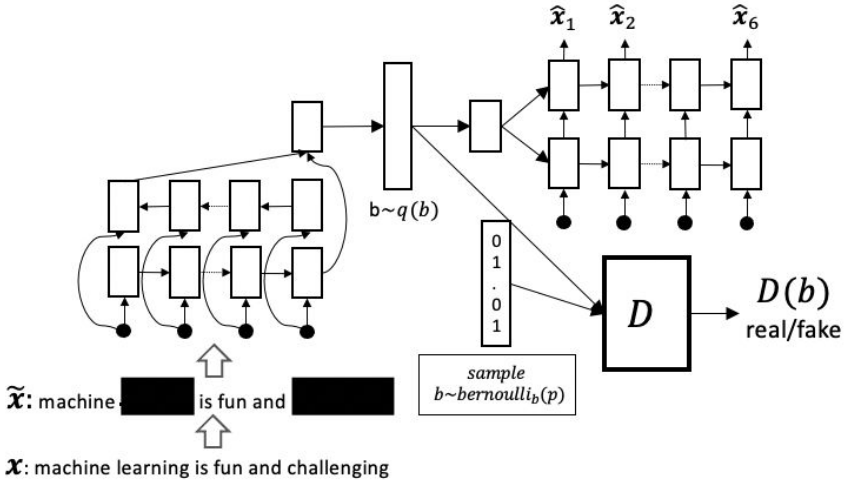
similar articles

2

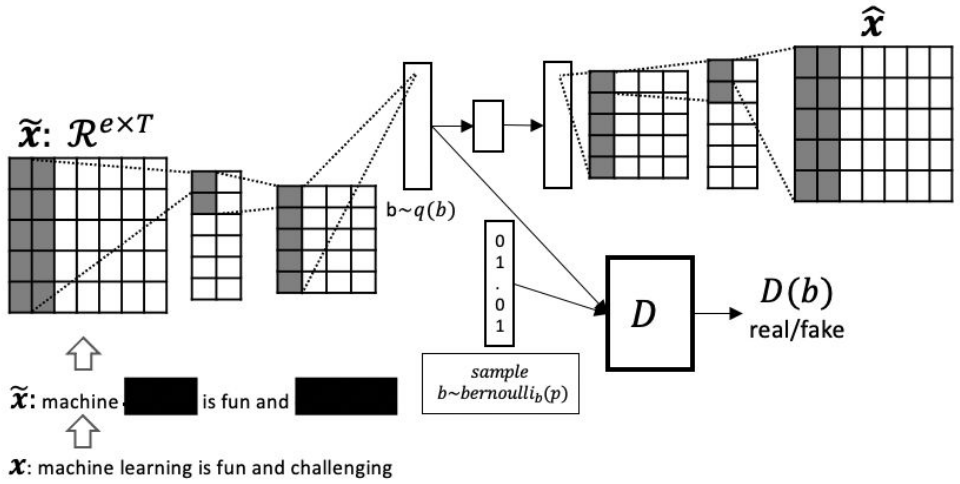


[Doan et al. 2021]

Adversarial Autoencoder Models

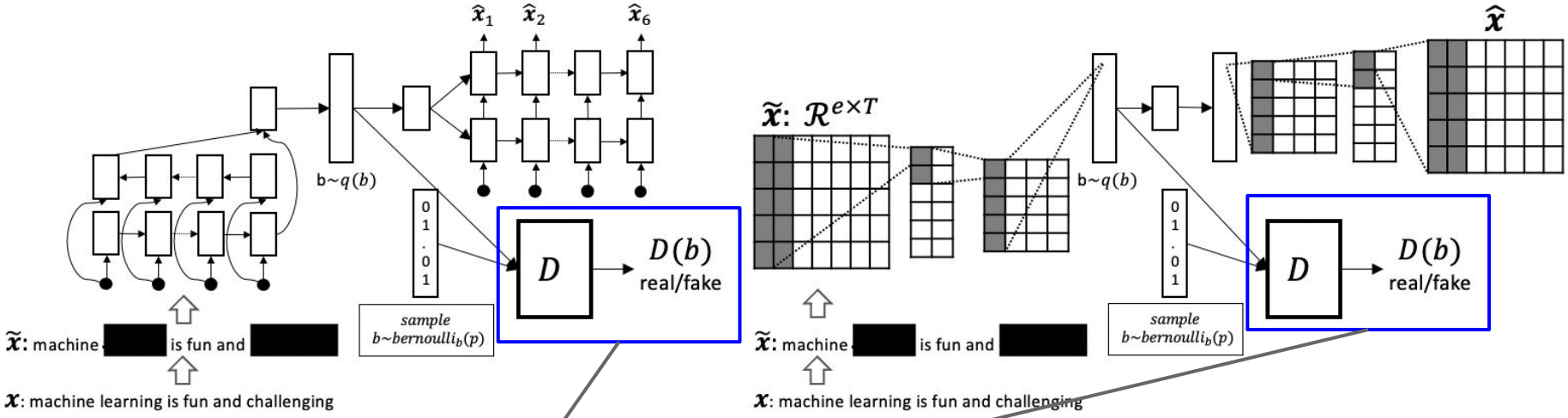


DABA-RNN



DABA-CNN

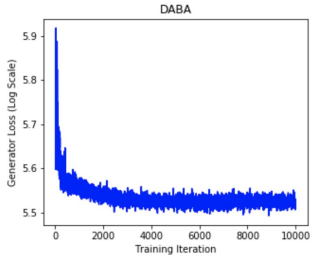
Adversarial Autoencoder Models



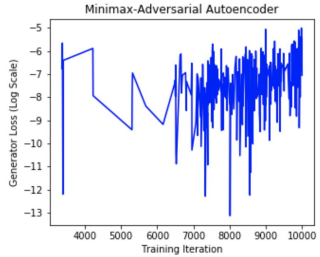
DABA-RNN **DABA-CNN**

$$\min_{\theta_E} \max_{M} \sum_{i=1}^{N} \sum_{j=1}^{N} M_{i,j} d^p_{\theta_G}(z_i, b_j)$$

$$\text{s.t.} \sum_{i=1}^{N} M_{i,j} = 1, \sum_{j=1}^{N} M_{i,j} = 1, M_{i,j} \in \{0, 1\}$$



Proposed Training



JSD Training

Text Retrieval Performance

Method	20Newsgroup				Reuters			
	P@10	P@50	P@100	MAP	P@10	P@50	P@100	MAP
ITQ-Linear	0.524	0.457	0.384	0.320	0.799	0.731	0.667	0.430
STH-Non-VAE	0.523	0.501	0.565	0.328	0.817	0.798	0.755	0.476
SemH-RBM	0.520	0.435	0.390	0.322	0.780	0.703	0.650	0.462
VDSH-VAE	0.552	0.550	0.431	0.339	0.839	0.815	0.775	0.495
NASH-VAE	0.579	0.544	0.539	0.349	**0.867**	0.840	0.799	0.501
OURS-MLP	0.559	0.542	0.565	0.339	0.836	0.831	0.772	0.497
OURS-RNN	0.628	0.579	0.565	**0.366**	0.859	0.833	**0.830**	**0.520**
OURS-CNN	**0.639**	**0.612**	**0.590**	0.341	0.860	**0.861**	0.820	0.509

(1) Significant improvement from linear and non-linear methods

(2) Representation Learning + Effective Quantization are necessary

Application in Image Retrieval

KL/JSD/Wasserstein Distance
- Inefficient to estimate

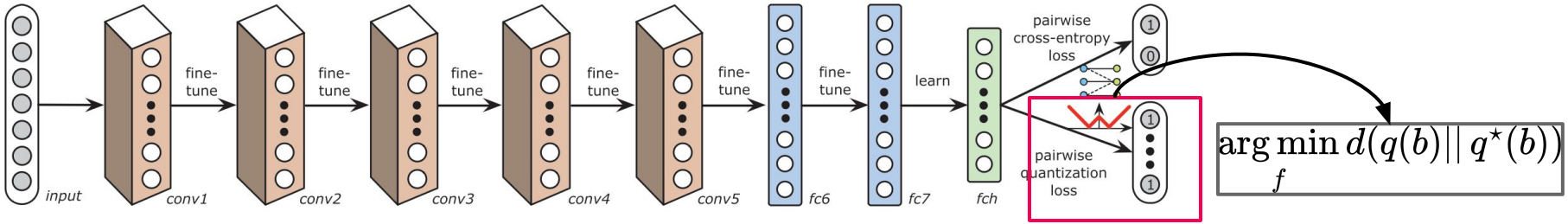
$$\mathcal{D}(\mu,\nu) = \left(\inf_{\gamma \in \Pi(\mu,\nu)} \int_{(z,b)\sim\gamma} p(z,b)||z-b||_2 dzdb \right)^{1/2}$$

Sliced Wasserstein Distance
- Several random projections are non-informative

$$\mathcal{D}(h(X),B) \approx \left(\frac{1}{L} \sum_{l=1}^{L} \mathcal{W}(\omega_l^T h(X), \omega_l^T B) \right)^{1/2}$$

Hash-Sliced Wasserstein Distance
Doan et al. 2022
- Fixed discriminative projections

$$\mathcal{D}(h(X),B) \approx \left(\frac{1}{m} \sum_{l=1}^{m} [\mathcal{W}(h(X)_{l,:}, B_{l,:})]^2 \right)^{1/2}$$



[Zhu et al. 2016]

Performance Evaluation (mAP)

Method	CIFAR-10			NUS-WIDE			COCO		
	16 bits	32 bits	64 bits	16 bits	32 bits	64 bits	16 bits	32 bits	64 bits
DSDH [40]	0.7909	0.8072	0.8278	0.8270	0.8455	0.8640	0.7331	0.7853	0.8074
DSDH-S	0.8187/**3.5%**	0.8439/**4.6%**	0.8517/**2.9%**	0.8282/**0.1%**	0.8461/**0.1%**	0.8712/**0.8%**	0.7330/*0.0%*	0.8030/**2.3%**	0.8404/**4.1%**
DSDH-C	0.8531/**7.9%**	0.8620/**6.8%**	0.8658/**4.6%**	0.8433/**2.0%**	0.8631/**2.1%**	0.8749/**1.3%**	0.7424/**1.3%**	0.8032/**2.3%**	0.8408/**4.1%**
HashNet [6]	0.6922	0.8311	0.8566	0.7728	0.8336	0.8654	0.6899	0.7666	0.8098
HashNet-S	0.8131/**17%**	0.8573/**3.2%**	0.8749/**2.1%**	0.8062/**4.3%**	0.8438/**1.2%**	0.8713/**0.7%**	0.7215/**4.6%**	0.7764/**1.3%**	0.8189/**1.1%**
HashNet-C	0.7939/**14%**	0.8467/**1.9%**	0.8691/**1.5%**	0.8002/**3.5%**	0.8437/**1.2%**	0.8791/**1.6%**	0.7202/**4.4%**	0.7789/**1.6%**	0.8202/**1.3%**
GreedyHash [50]	0.8223	0.8474	0.8646	0.7802	0.8081	0.8328	0.6533	0.7219	0.7561
GreedyHash-S	0.8280/**0.7%**	0.8497/**0.3%**	0.8653/**0.1%**	0.7815/**0.1%**	0.8083/*0.0%*	0.8390/**0.7%**	0.6668/**2.1%**	0.7291/**1.0%**	0.7618/**0.8%**
GreedyHash-C	0.8375/**1.9%**	0.8536/**0.7%**	0.8722/**0.9%**	0.7890/**1.1%**	0.8179/**1.2%**	0.8477/**1.8%**	0.6637/**1.6%**	0.7299/**1.1%**	0.7712/**2.0%**
DCH [5]	0.8302	0.8432	0.8558	0.8015	0.8061	0.8040	0.7578	0.7792	0.7723
DCH-S	0.8372/**0.8%**	0.8515/**1.0%**	0.8602/**0.5%**	0.8058/**0.5%**	0.8079/**0.2%**	0.8067/**0.3%**	0.7657/**1.1%**	0.7831/**0.5%**	0.7803/**1.0%**
DCH-C	0.8446/**1.7%**	0.8596/**1.9%**	0.8711/**1.8%**	0.8159/**1.8%**	0.8145/**1.0%**	0.8155/**1.4%**	0.7702/**1.6%**	0.7892/**1.3%**	0.7807/**1.1%**
CSQ [58]	0.8069	0.8291	0.8366	0.7992	0.8384	0.8596	0.6783	0.7550	0.8146
CSQ-S	0.8401/**4.1%**	0.8555/**3.2%**	0.8554/**2.3%**	0.8044/**0.7%**	0.8495/**1.3%**	0.8626/**0.4%**	0.7036/**3.7%**	0.7765/**2.8%**	0.8234/**1.0%**
CSQ-C	0.8457/**4.8%**	0.8558/**3.2%**	0.8652/**3.4%**	0.8054/**0.8%**	0.8511/**1.5%**	0.8701/**1.2%**	0.6989/**3.0%**	0.7752/**2.7%**	0.8255/**1.3%**
DBDH [60]	0.7660	0.8223	0.8492	0.8305	0.8552	0.8666	0.7202	0.7826	0.8042
DBDH-S	0.8458/**10%**	0.8587/**4.4%**	0.8603/**1.3%**	0.8387/**1.0%**	0.8577/**0.3%**	0.8680/**1.8%**	0.7461/**2.2%**	0.7996/**3.7%**	0.8336/**4.3%**
DBDH-C	0.8466/**10%**	0.8593/**4.5%**	0.8668/**2.1%**	0.8395/**1.1%**	0.8633/**0.9%**	0.8760/**1.1%**	0.7389/**2.6%**	0.7889/**0.8%**	0.8308/**3.9%**

Performance Evaluation (mAP)

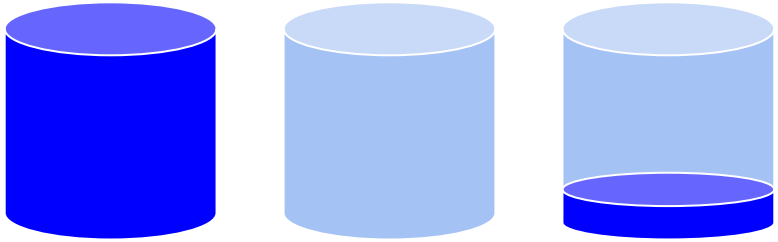
Method	CIFAR-10			NUS-WIDE			COCO		
	16 bits						bits	64 bits	
DSDH [40]	0.7909						7853	0.8074	
DSDH-S	0.8187/3						8030/**2.3%**	0.8404/**4.1%**	
DSDH-C	0.8531/7						8032/**2.3%**	0.8408/**4.1%**	
HashNet [6]	0.6922						7666	0.8098	
HashNet-S	0.8131/1						7764/**1.3%**	0.8189/**1.1%**	
HashNet-C	0.7939/1						7789/**1.6%**	0.8202/**1.3%**	
GreedyHash [50]	0.8223						7219	0.7561	
GreedyHash-S	0.8280/0						7291/**1.0%**	0.7618/**0.8%**	
GreedyHash-C	0.8375/1						7299/**1.1%**	0.7712/**2.0%**	
DCH [5]	0.8302						7792	0.7723	
DCH-S	0.8372/0						7831/**0.5%**	0.7803/**1.0%**	
DCH-C	0.8446/1						7892/**1.3%**	0.7807/**1.1%**	
CSQ [58]	0.8069						7550	0.8146	
CSQ-S	0.8401/4						7765/**2.8%**	0.8234/**1.0%**	
CSQ-C	0.8457/4						7752/**2.7%**	0.8255/**1.3%**	
DBDH [60]	0.7660						7826	0.8042	
DBDH-S	0.8458/**10%**	0.8587/**4.4%**	0.8603/**1.3%**	0.8387/**1.0%**	0.8577/**0.3%**	0.8680/**1.8%**	0.7461/**2.2%**	0.7996/**3.7%**	0.8336/**4.3%**
DBDH-C	0.8466/**10%**	0.8593/**4.5%**	0.8668/**2.1%**	0.8395/**1.1%**	0.8633/**0.9%**	0.8760/**1.1%**	0.7389/**2.6%**	0.7889/**0.8%**	0.8308/**3.9%**

Table: Averaged running time per epoch across different supervised hashing methods (in seconds).

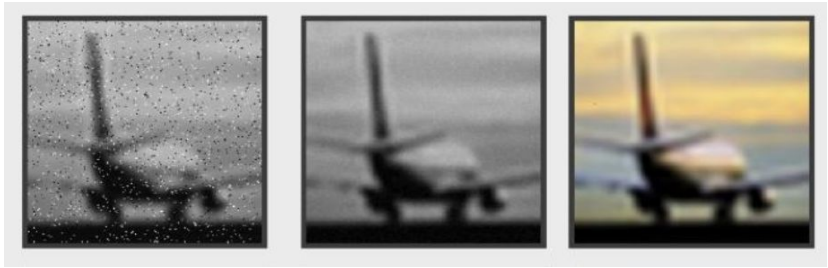
Dataset	Original	SWD	HSWD
CIFAR-10	19.4	24.2	17.1/**40%**
NUS-WIDE	58.3	71.2	50.1/**41%**
COCO	55.6	68.1	49.5/**37%**

More computationally efficient than the original quantizations

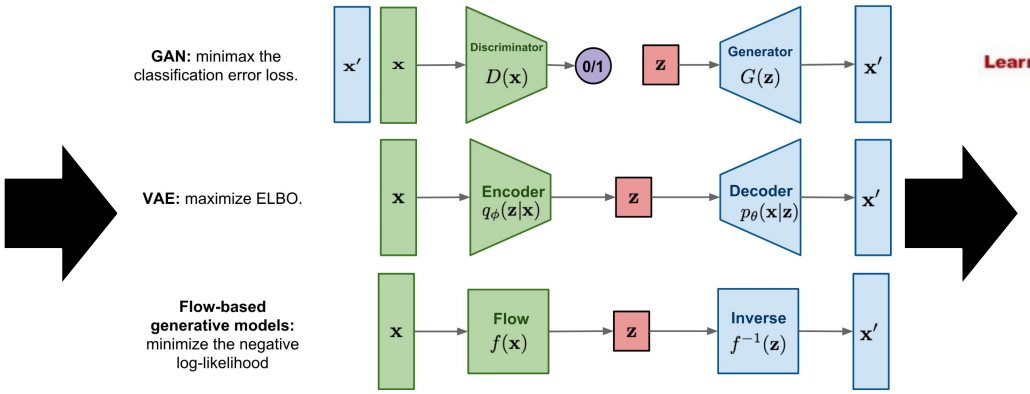
Learning hash function in real applications



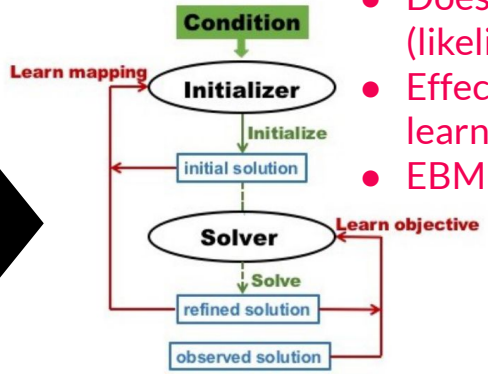
Fully Labeled | Fully Unlabeled | Partially Unlabeled



Corruption and Distributional Drift



Source: Lil'Log

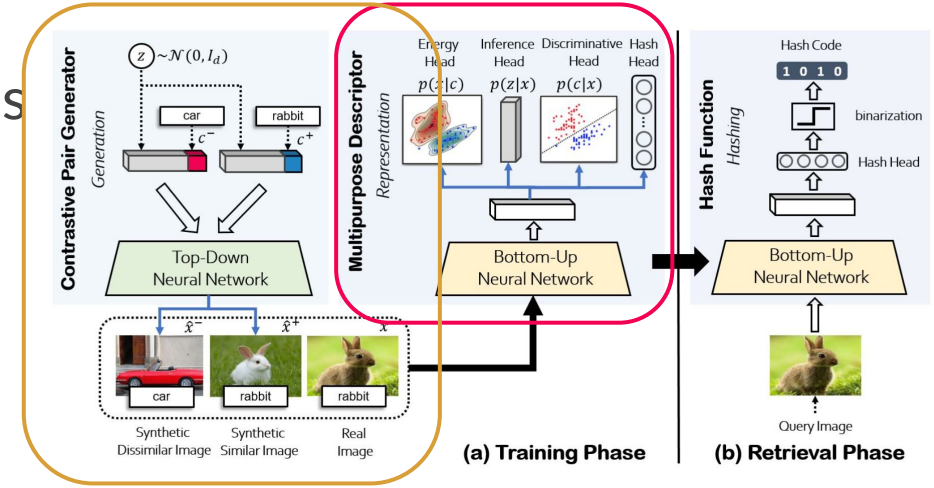


Cooperative Generative EBM [Xie et al. 2018]

- Does not have mode collapse (likelihood learning)
- Effective representation learning
- EBM is more robust

Multipurpose Generative Hashing Network

- Learns to solve multiple objectives
  - **Energy head** to estimate p(x,c) and to generate new data for hash learning.
  - **Classification head** to estimate p(c|x).
  - **Hashing head** to learn hash function: with contrastive generated samples.



- **Learning**: short-run MCMC

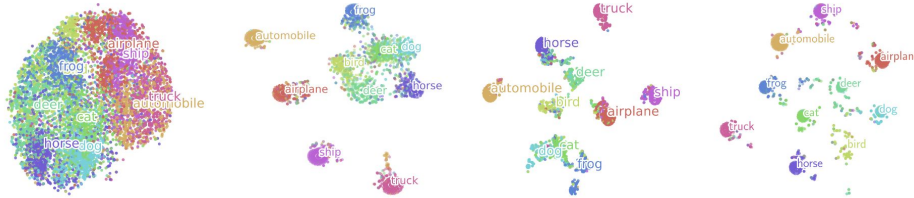
$$\hat{x} = g(c, \hat{z}; \Lambda), \hat{z} \sim \mathcal{N}(0, I_d)$$

$$\tilde{x}_{t+1} = \tilde{x}_t - \frac{\delta^2}{2} \frac{\partial f_E(\tilde{x}_t, c; \Theta_E)}{\partial \tilde{x}} + \delta \mathcal{N}(0, I_D), \tilde{x}_0 = \hat{x}$$

generator's samples

Cooperative EBM Learning

[Doan et al. 2021



Retrieval performance and robustness

Retrieval Performance (mAP)

Method	NUS-WIDE		CIFAR10	
	32-bit	64-bit	32-bit	64-bit
ITQ-*Unsupervised*	0.405	0.373	0.414	0.449
CNNH-*Linear*	0.583	0.593	0.472	0.489
DVStH-*VAE*	0.680	0.698	0.695	0.708
Hashnet-*Nonlinear*	0.699	0.711	0.699	0.711
HashGAN	0.737	0.744	0.731	0.735
OURS	**0.759**	**0.778**	**0.742**	**0.781**

(1) Both GAN-based (HashGAN) and ours achieve significant improvement over existing linear and non-linear supervised methods

(2) Our method achieve 2-5% improvement over HashGAN (GAN-based data synthesis)

Retrieval performance and robustness

Out-of-distribution (OOD) Queries (mAP)

Train with SVHN	HashNet	HashGAN	OURS
Query: MNIST	0.181	0.354	**0.609**
Query: SVHN	0.837	0.889	**0.895**
Train with MNIST	**HashNet**	**HashGAN**	**OURS**
Query: SVHN	0.193	0.280	**0.498**
Query: MNIST	0.957	0.990	**0.991**

(1) Significant performance drop when the query is out-of-distribution data in both SOTA non-linear method & HashGAN

(2) Our method still perform reasonably well in OOD retrieval (**2x better**), compared to other methods.

Retrieval performance and robustness

Input Data Corruption (mAP)

Salt-and-Pepper Corruption			
	HashNet	HashGAN	OURS
Clean	0.837	0.889	**0.895**
Corrupted	0.181	0.354	**0.609**
Large Rectangular Mask Corruption			
	HashNet	HashGAN	OURS
Clean	0.957	0.990	**0.991**
Corrupted	0.193	0.280	**0.498**

(1) Significant performance drop when the data is corrupted in other methods.

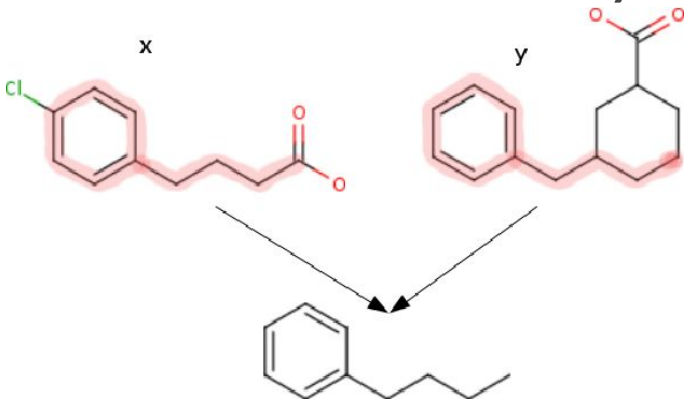
(2) Our method still perform reasonably well with corrupted data (**almost 2x better**), compared to other methods.

How similar are two graphs?

Intrusion detection

Behavioral patterns

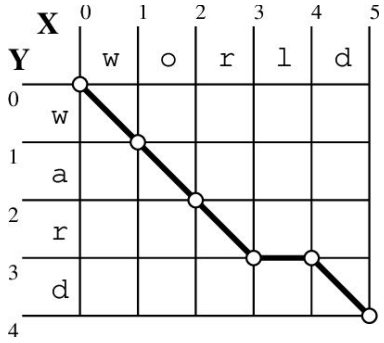
Molecule similarity



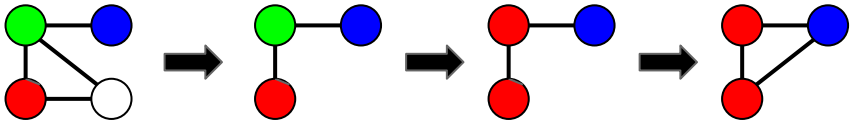
How similar are two graphs?

- Application-agnostic measures such as **GED** or **MCS** are important.
  - MCS is a special case of GED

- Explicitly explainable
  - In case of comparing molecules, GED tells the sequence of atoms/bonds to insert/delete or substitute that transforms once molecule to the other.



String edit distance is based on the sequence of edits (character insertion, deletion or substitution) to transform one string to the other.

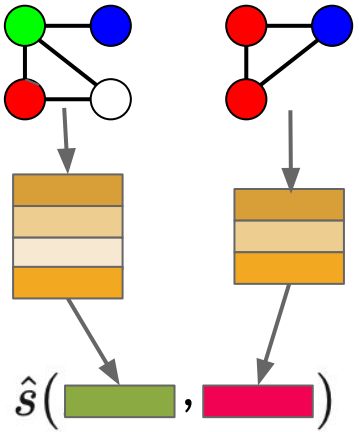


Graph Edit Distance is based on the sequence of edits (node/edge insertion, deletion or substitution) to transform one graph to the other.

Existing GED Computation Solutions

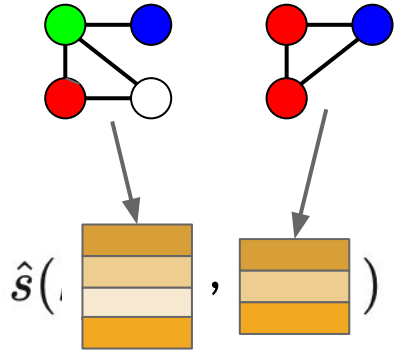
- Exact GED computation is NP-hard
  - Combinatorial Search: A*
  - Impractical for graphs with more than a few tens of nodes.

Sub-optimal Performance



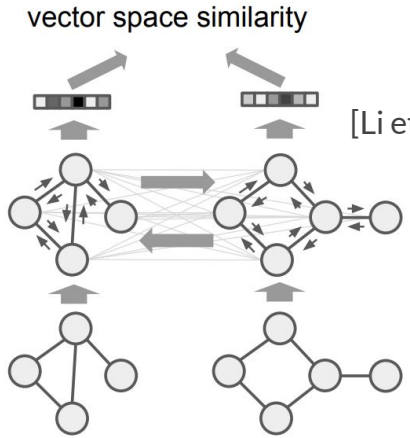
Function of graph embeddings

- Heuristic Order
- Does not capture, thus cannot explain the edit operations.



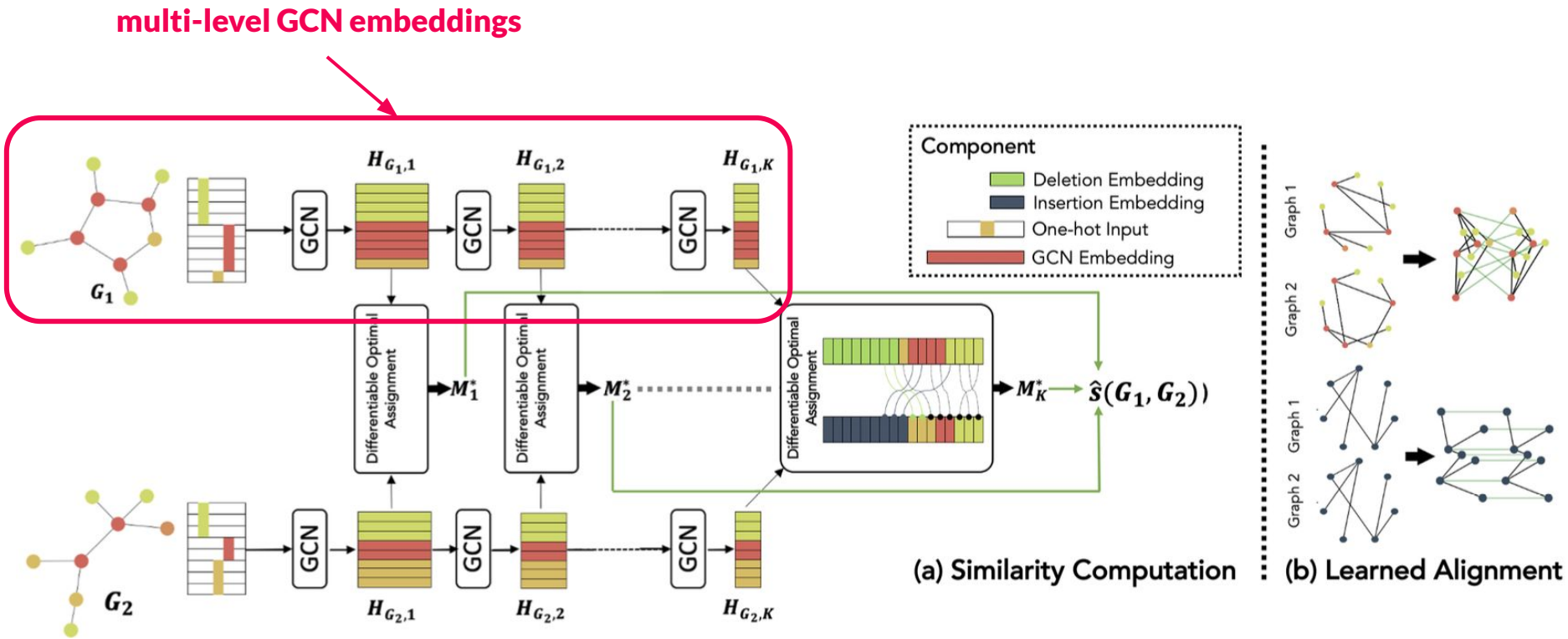
Function of node embeddings

Graph-matching do not learn bijective mapping

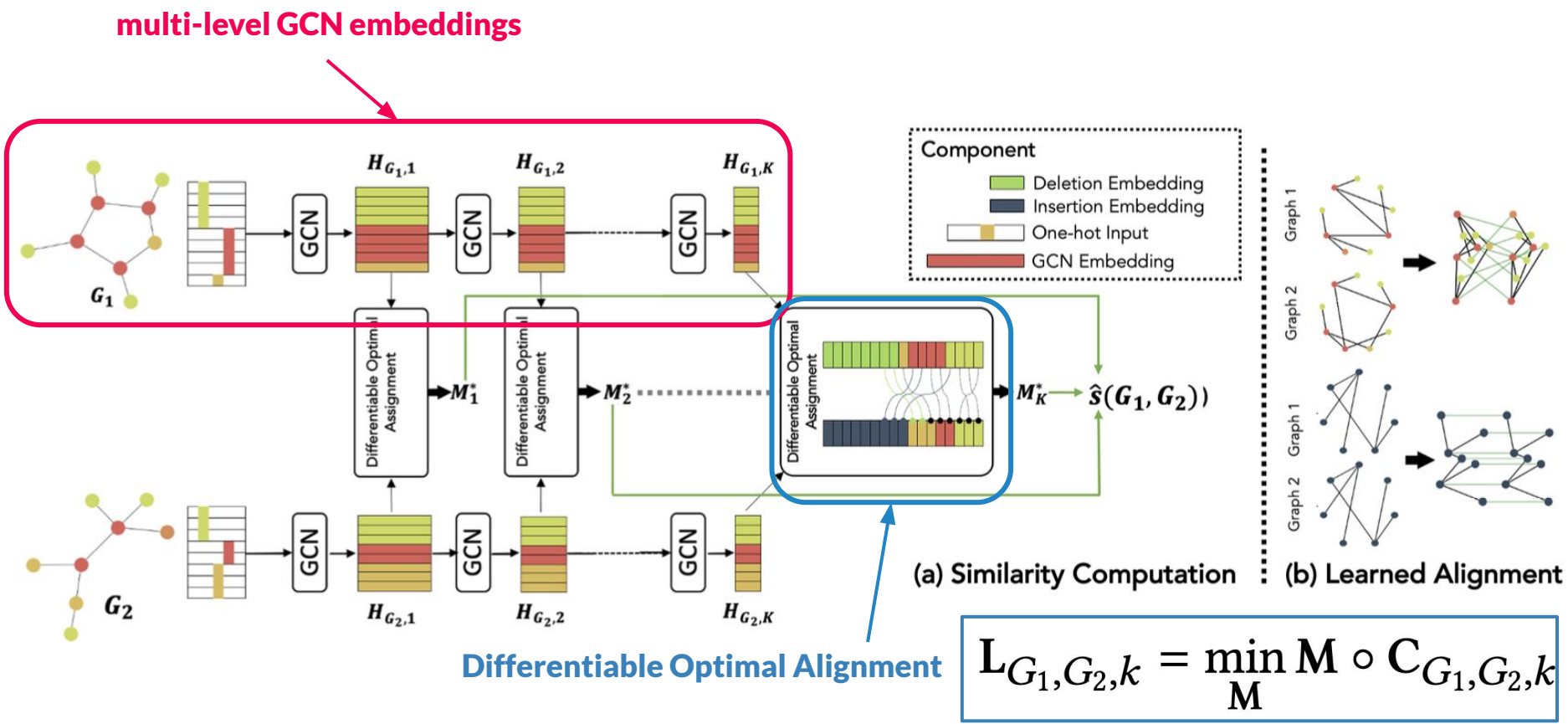


[Li et al. 2019]

Graph Optimal Transport Similarity Computation



Graph Optimal Transport Similarity Computation



$$\mathrm{L}_{G_1,G_2,k} = \min_{\mathbf{M}} \mathbf{M} \circ \mathrm{C}_{G_1,G_2,k}$$

Augmented similarity matrix

node assignment

node deletion

$$\mathbf{C}_{G_1,G_2,k} = \left[\begin{array}{ccc|ccc} \mathbf{c}_{k,1,1} & \cdots & \mathbf{c}_{k,1,N_2} & \mathbf{d}_{k,1} & \cdots & \infty \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ \mathbf{c}_{k,N_1,1} & \cdots & \mathbf{c}_{k,N_1,N_2} & \infty & \cdots & \mathbf{d}_{k,N_1} \\ \hline \mathbf{a}_{k,1} & \cdots & \infty & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ \infty & \cdots & \mathbf{a}_{k,N_2} & 0 & \cdots & 0 \end{array}\right]$$

node addition

$$\mathbf{c}_{k,i,j} = \mathbf{c}(\mathbf{h}_{G_1,k,i}, \mathbf{h}_{G_2,k,j})$$

$$\mathbf{a}_{k,i} = c(\mathbf{h}_{G_2,k,i}, \mathbf{h}_{k,a})$$

$$\mathbf{d}_{k,i} = c(\mathbf{h}_{G_1,k,i}, \mathbf{h}_{k,d})$$

global embedding vectors

M is now a permutation matrix

➡ Simpler and more stable derivative

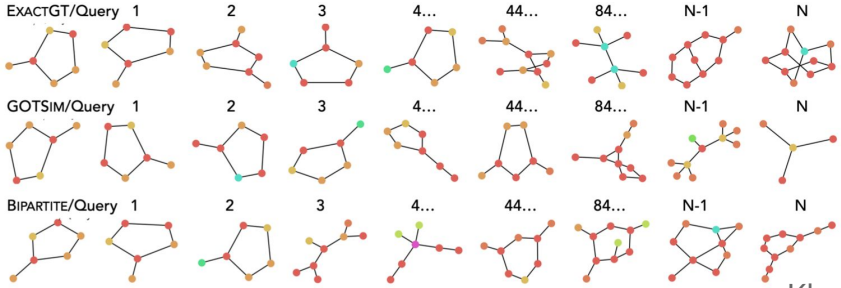
$$\frac{\partial \mathbf{L}_{G_1,G_2,k}}{\partial \theta} = \mathbf{M}_k^* \frac{\partial \mathbf{C}_{G_1,G_2,k}}{\partial \theta}$$

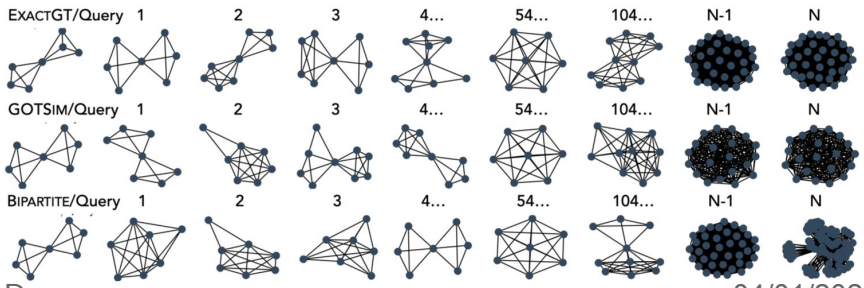
SOTA Retrieval Performance

Exact Ground-truth available

Only approximated ground-truth available

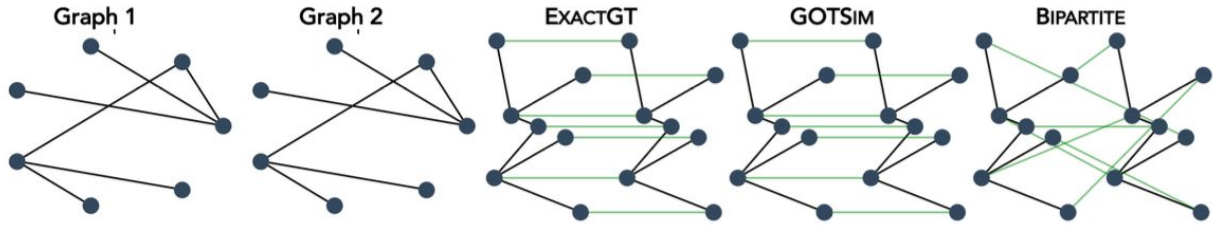
	AIDS			LINUX			PTC		
	τ	ρ	P@10	τ	ρ	P@10	τ	ρ	P@10
ExactGT	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	–	–	–
Bipartite	0.26 ± 0.02	0.37 ± 0.02	0.32 ± 0.04	0.32 ± 0.02	0.43 ± 0.03	0.45 ± 0.03	0.65 ± 0.02	0.82 ± 0.02	0.79 ± 0.03
Hausdorff	0.48 ± 0.02	0.63 ± 0.03	0.54 ± 0.05	0.78 ± 0.01	0.88 ± 0.01	0.78 ± 0.05	0.78 ± 0.01	0.91 ± 0.01	0.90 ± 0.02
EmbMean	0.41 ± 0.04	0.53 ± 0.04	0.60 ± 0.10	0.55 ± 0.02	0.60 ± 0.02	0.51 ± 0.01	0.16 ± 0.04	0.23 ± 0.06	0.41 ± 0.08
EmbMax	0.42 ± 0.02	0.57 ± 0.01	0.61 ± 0.06	0.57 ± 0.03	0.65 ± 0.01	0.71 ± 0.01	0.21 ± 0.03	0.29 ± 0.03	0.49 ± 0.03
EmbGated	0.43 ± 0.02	0.66 ± 0.02	0.70 ± 0.04	0.58 ± 0.03	0.88 ± 0.01	0.81 ± 0.01	0.66 ± 0.10	0.81 ± 0.13	0.84 ± 0.08
GMN	0.47 ± 0.05	0.69 ± 0.02	0.72 ± 0.02	0.78 ± 0.03	0.88 ± 0.01	0.80 ± 0.01	0.40 ± 0.02	0.71 ± 0.03	0.70 ± 0.04
SimGNN	0.67 ± 0.03	0.82 ± 0.02	0.84 ± 0.04	0.80 ± 0.01	0.92 ± 0.01	0.82 ± 0.05	0.80 ± 0.02	0.93 ± 0.01	0.91 ± 0.01
GraphSim	0.68 ± 0.03	0.57 ± 0.03	0.76 ± 0.03	0.83 ± 0.02	0.92 ± 0.02	0.84 ± 0.03	0.79 ± 0.01	0.91 ± 0.01	0.91± 0.02
GOTSim	**0.72 ± 0.02**	**0.86± 0.02**	**0.87± 0.03**	**0.89 ± 0.02**	**0.92± 0.01**	**0.86± 0.02**	**0.82 ± 0.01**	**0.95± 0.01**	**0.94± 0.01**



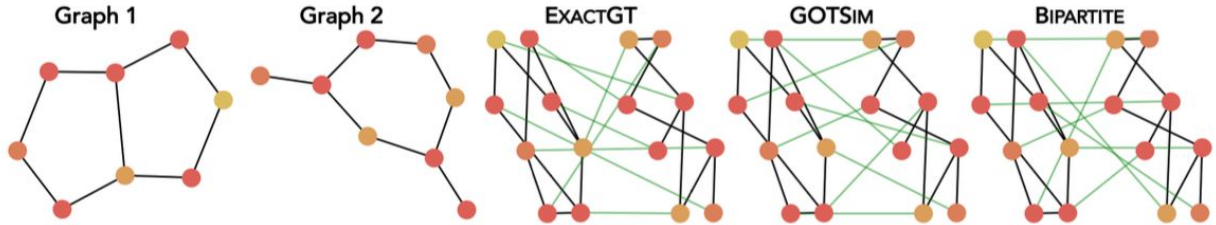


Learned Node Assignment (GED approx.)

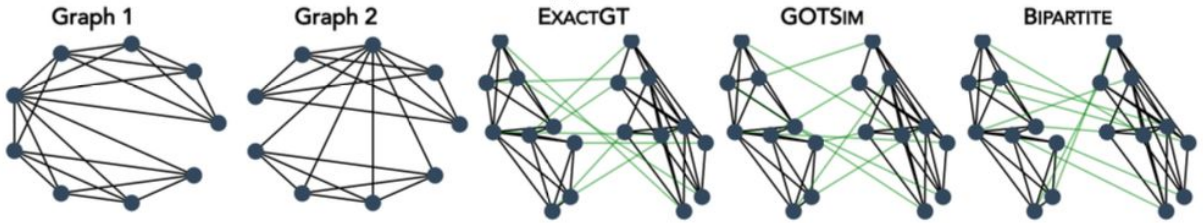
LINUX



PTC



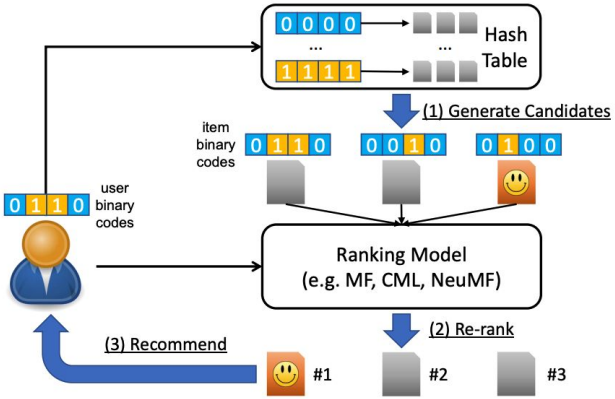
IMDB



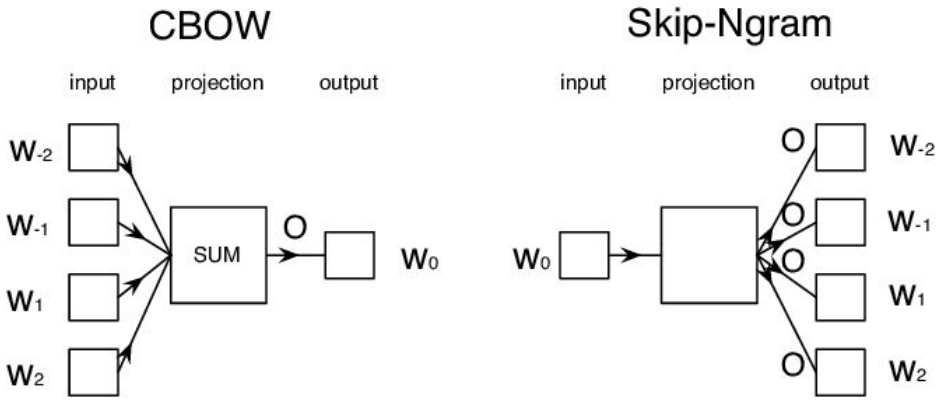
Capture the exact alignment of isomorphic graphs

Capture similar alignments as those of the ground-truth GED computation, for both multi-labeled and unlabeled cases

Retrieval in Machine Learning



Real-time Recommendation
(Kang et al. 2019)



Dynamic Negative Sampling
(Chen et al. 2018)

References

Henzinger et al. *Finding near-duplicate web pages: a large-scale evaluation of algorithms*. SIGIR 2006.

Salakhutdinov et al. *Semantic hashing*. IJAR 2009.

Weiss et al. *Spectral hashing*. NIPS 2009.

Li et al. *Hashing algorithms for large-scale learning*. NIPS 2011.

Li et al. *Hashing algorithms for large-scale learning*. NIPS 2011.

Gong et al. *Iterative quantization: A procrustean approach to learning binary codes for large-scale image retrieval*. TPAMI 2012.

Li et al. *Coding for random projections*. ICML 2014.

Shrivastava et al. *Asymmetric LSH (ALSH) for Sublinear Time Maximum Inner Product Search (MIPS)*. NIPS 2014.

Zhu et al. *Deep Hashing Network for Efficient Similarity Retrieval*. AAAI 2016.

Chen et al. Improving Negative Sampling for Word Representation using Self-embedded Features. WSDM 2018.

Xie et al. *Cooperative Training of Descriptor and Generator Networks*. TPAMI 2018.

Doan et al. *Adversarial factorization autoencoder for look-alike modeling*. CIKM 2019.

Kang et al. *Candidate generation with binary codes for large-scale top-n recommendation*. CIKM 2019.

Li et al. *Graph matching networks for learning the similarity of graph structured objects*. ICML 2019.

Johnson et al. Billion-scale similarity search with GPUs. IEEE Transactions on Big Data 2019.

Chen et al. *A Simple Framework for Contrastive Learning of Visual Representations*. ICML 2020.

Tan et al. *Fast item ranking under neural network based measures*. WSDM 2020.

References

Doan et al. *Interpretable graph similarity computation via differentiable optimal alignment of node embeddings*. SIGIR 2021.

Doan et al. Cooperative Learning of Energy-Based Generative Hashing Networks. 2021.

Doan et al. *One Loss for Quantization: Deep Hashing with Discrete Wasserstein Distributional Matching*. CVPR 2022.

Credits to my collaborators for this talk!

Chandan Reddy
Virginia Tech

Keerthi Selvaraj
Linkedin AI

Ping Li
Baidu Research

Saurav Manchanda
Instacart

Sarkhan Badirli
Eli Lilly

Jianwen Xie
Baidu Research

Shulong Tan
Baidu Research

Weijie Zhao
RIT

Peng Yang
Baidu Research

and others …

THANK YOU!

Contact: khoadoan@vt.edu / doankhoadang@gmail.com
Website: https://khoadoan.me