Approximate Query Processing

- Space Partitioning-based
  - Tree
  - Encoding
  - Locality Sensitive Hashing
- Graph-based Methods

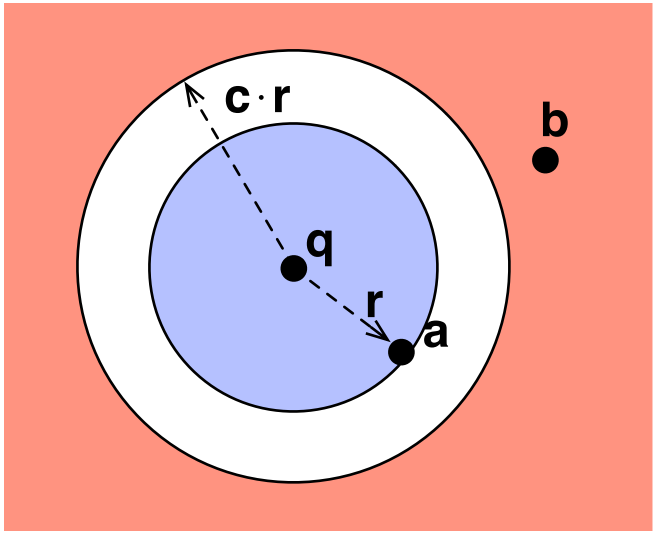
Notes:

- Recent works mainly in the Database area
- Prefer ease of exposition over rigor
- Categorization is not fixed/unique

Locality Sensitive Hashing (LSH)

- From the perspective of <u>collision probability</u>
  - (Ordinary) hash function h:
    - <u>Pr[h(x) = h(y)]</u> = ε, if x $\neq$ y

c.f., Cryptographic hash functions
Pr [h(x) = h(y)] = 2^{-m}, if
Hamming(x, y) = 1



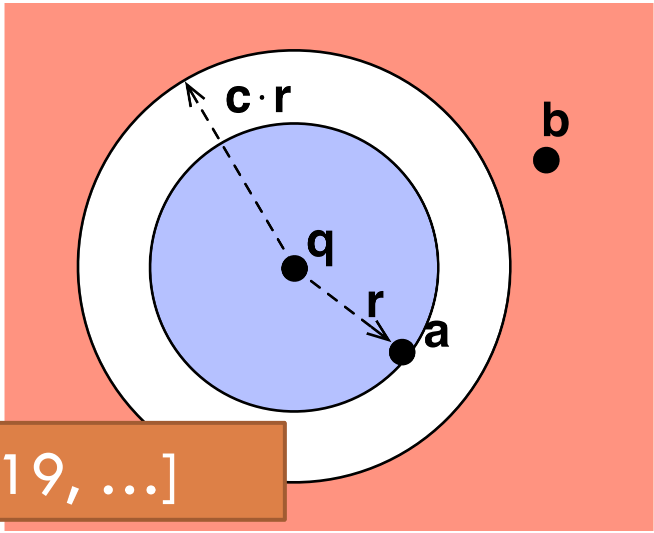
Locality Sensitive Hashing (LSH)

- From the perspective of <u>collision probability</u>
  - (Ordinary) hash function h:
    - Pr[h(x) = h(y)] = ε, if x ≠ y
  - LSH
    - Pr[h(x) = h(y)] increases with locality
    - Randomness comes from r.v. h ∈ H

c.f., Cryptographic hash functions
Pr [h(x) = h(y)] = 2^{-m}, if
Hamming(x, y) = 1

(r_1, r_2, p_1, p_2)-sensitive [IM98]
- Pr[h(x) = h(y)] ≥ p_1 , if dist(x, y) ≤ r_1
- Pr[h(x) = h(y)] ≤ p_2 , if dist(x, y) ≥ r_2

Pr[h(x) = h(y)] = sim(x, y) [C02] too narrow



Can be generalized [SWQZ+14, ACPS18, CKPT19, …]

Locality Sensitive Hashing (LSH)

- Equality search
  - Index: store o into bucket h(o)
  - Query:
    - retrieve every o_i in the bucket h(q)
    - verify if $o_i = q$

Pr[h(q) = h(o)] = 1/B

LSH is the **best** approach using **sub-quadratic** space

Locality Sensitive Hashing (LSH)

- Equality search
  - Index: store o into bucket h(o)
  - Query:
    - retrieve every o_i in the bucket h(q)
    - verify if $o_i = q$

Pr[h(q) = h(o)] = 1/B

- LSH
  - ∀h ∈ *LSH-family*, **Pr**[Q(h(q)) = Q(h(o))] = f(Dist(q, o))
    - Q( ): quantization (not essential)
    - "Near-by" points have more chance of colliding with q than "far-away" points
  - Similar index & query procedures, with a weak probabilistic guarantee

c.f., [PIM12] for the rigorous QP procedure

➔ Repeat to boost the guarantee

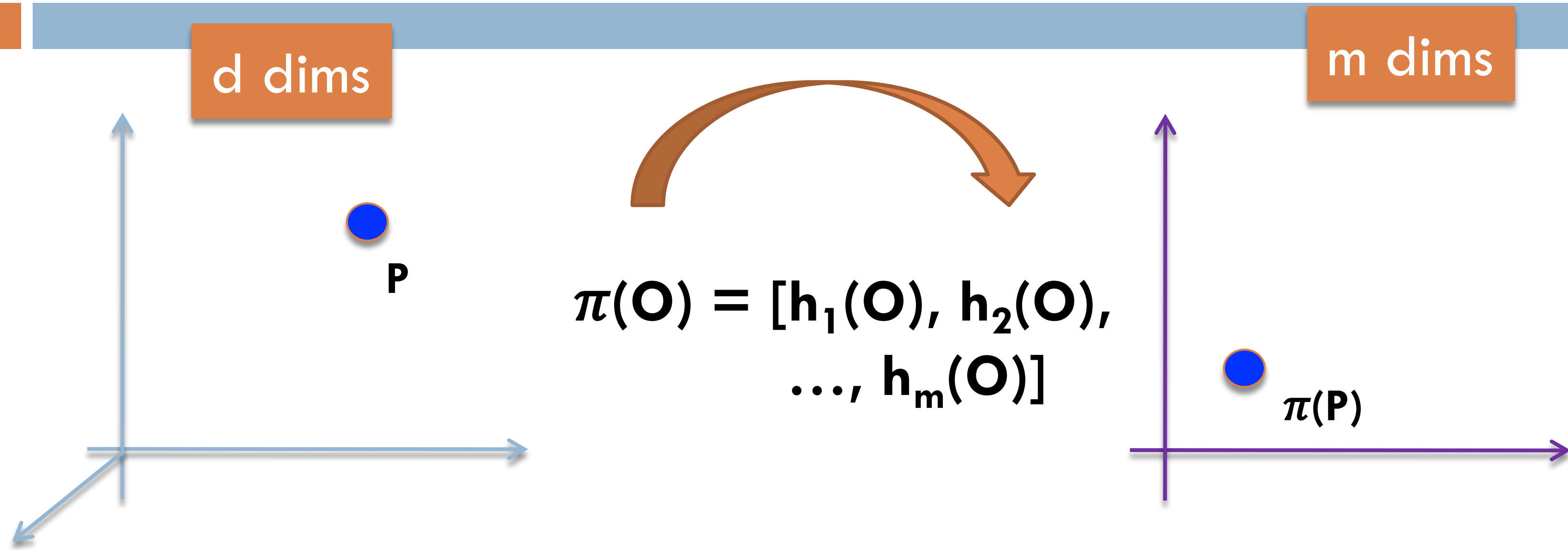
LSH Families

- Many are known
  - L_p ($0 < p \le 2$): use p-stable distribution to generate the projection vector
    - For L_2, just use random Gaussian vector
    - Other families exists, e.g., sparse random projection
  - Angular distance (arccos): SimHash
  - Jaccard: minhash (based on random permutation)
  - Hamming:
    - random projection
    - covering LSH

Comments

- New queries can be reduced to known LSH cases
  - Maximum inner product search (MIPS)
  - Set containment
  - Group aggregated query
- Related to various distortion-bounded embedding
  - Edit distance: CGK-embedding to Hamming with O(K) distortion

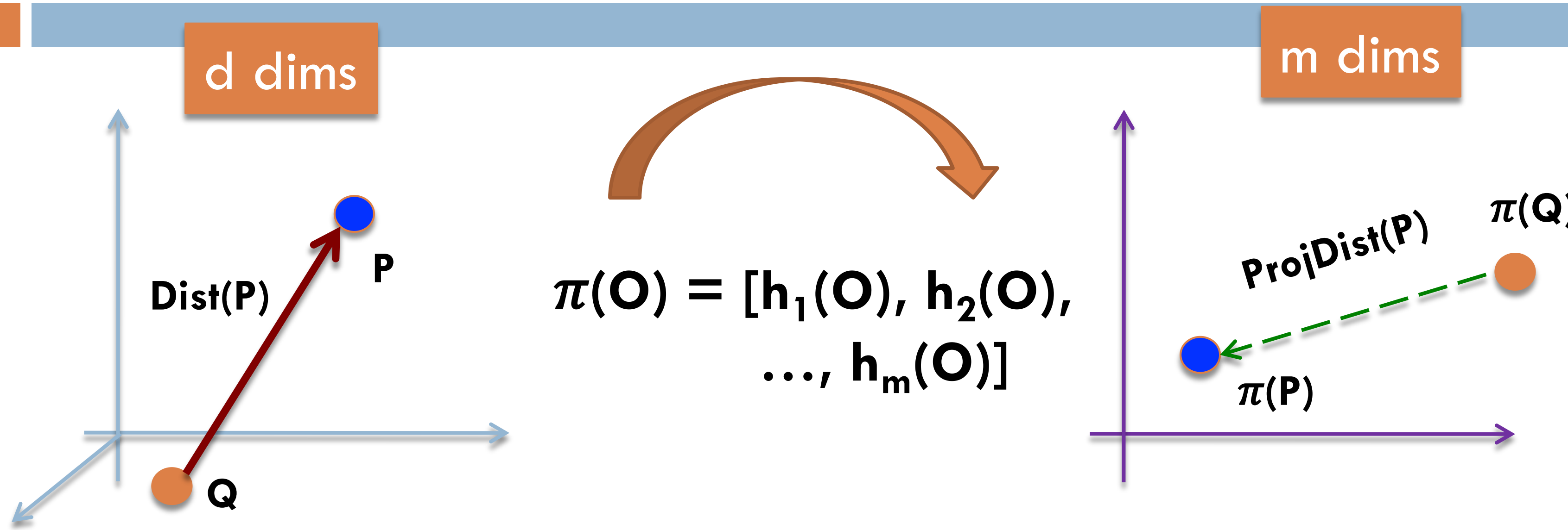
Probabilistic Mapping



$$\pi(O) = [h_1(O), h_2(O), \ldots, h_m(O)]$$

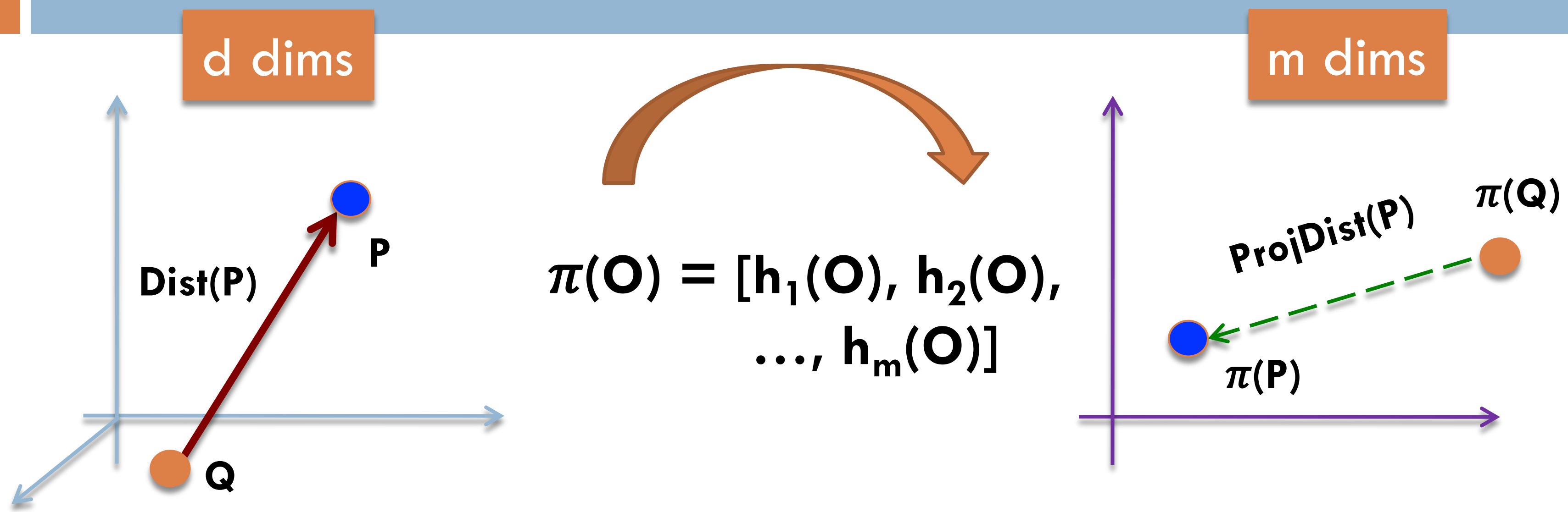
- Probabilistic, linear mapping from the original space to the projected space

Probabilistic Mapping



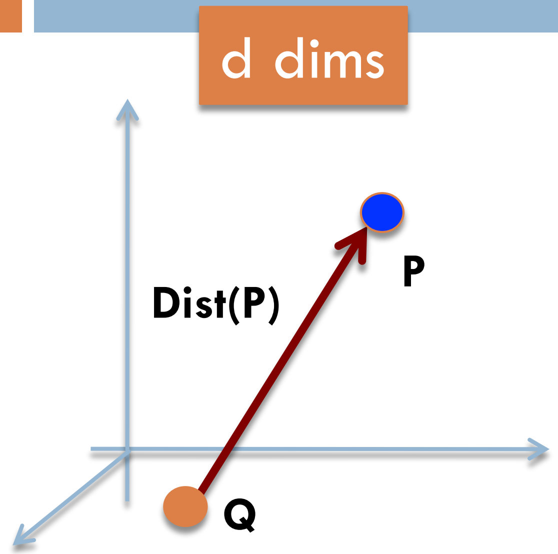
- Probabilistic, linear mapping from the original space to the projected space
- What about the distances (wrt Q or π(Q)) in these two spaces?

Probabilistic Distance Tracking Property of the Mapping

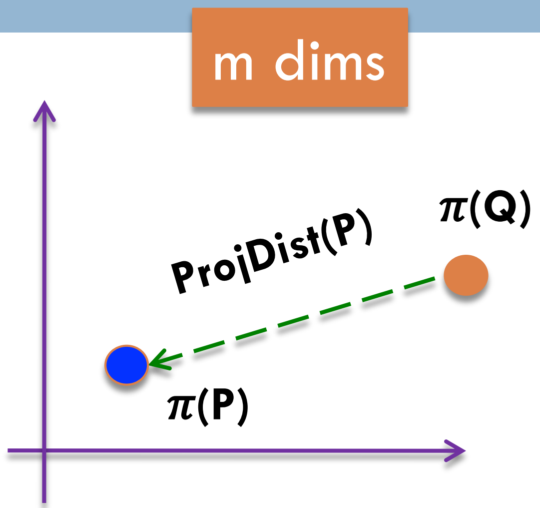


- $ProjDist(P)^2 \sim Dist(P)^2 * \chi^2_m$ [SWQZ+14]
  - $ProjDist(P)^2$ can be computed (incrementally) from $h_i(P)$ and $h_i(Q)$ due to the linearity of the hash function
  - Can be generalized to other p-stable LSH functions

Probabilistic Distance Tracking Property of the Mapping



$$\pi(O) = [h_1(O), h_2(O), \ldots, h_m(O)]$$



LSH provides a probabilistic distance-preserving mapping between the two spaces

Johnson & Lindenstrauss Lemma [JL84] only works for L2 and induces a method that requires more space than LSH [AIR18]

Roadmap

- Roadmap
  - Practical LSH methods (i.e., linear index complexity)
  - Data-dependent LSH methods

Practical Variations of LSH

- Easy to relax the LSH method in practice at the cost of no worst-case guarantees
  - E2LSH: use fewer number of random projections
  - Multiprobe LSH (entropyLSH and other variants): space-time tradeoff
  - LSH in practice: use empirically tuned parameters (k, l)
  - SRS: a space-saving LSH index with in-memory or disk implementations
  - QALSH: use multiple B-trees for the index and use collision counting strategy based on LSH
  - HD-index: space filling curves as pseudo-LSH functions
  - SK-LSH: Replace LSB-tree/forest by a dimension-wise linear mapping
  - …

Data-sensitive Hashing

- LSH is data-insensitive
  - Indexing hyper-parameters determined by the shape of the data only
  - Indexing parameters are randomly generated
- Efforts to make data-sensitive, LSH-like methods
  - [AR15]
    - Aim: break the lower bounds of ρ
  - DSH
  - OPFA / NeOPFA
- Learning-to-hash methods
  - NSH [PCM15]
  - [LYZX+18] and many in the ML/CV communities

c.f., [AIR18]

c.f., https://learning2hash.github.io and https://cs.nju.edu.cn/lwj/L2H.html

DSH [GJLO14]

- Learn a family of (hash) functions, H, that preserves kNN of queries
  1. Training data: ⇒ $$\mathbf{W}_{ij} = \begin{cases} 1 & \text{, if } o_j \in kNN(q_i) \\ -1 & \text{, if } o_j \notin kNN(q_i) \wedge o_j \text{ is sampled} \\ 0 & \text{otherwise.} \end{cases}$$
     - sampled queries
     - their k-NN objects (+ve)
     - samples non-c*k-NN objects (-ve)
  2. Function family:
     - Thresholded linear functions $h(\mathbf{x}; \mathbf{a}) = \operatorname{sgn}(\mathbf{a}^\top \mathbf{x})$
  3. Learn one hash function

$$\arg\min_h \sum_i \sum_j \ell(q_i, o_j)\, \mathbf{W}_{ij} \quad \text{, where } \ell(q, o) = (h(q) - h(o))^2$$

DSH [GJLO14]

- Learn a family of (hash) functions, H, that preserves kNN of queries
  4. Learn multiple hash functions
     - Multiplicative updates on W_{ij}
       - Increase W_{ij} if incorrectly classified
       - Decrease W_{ij} if correctly classified
     - (Under some assumptions) obtain H that satisifies the (k, ck, p_1, p_2)-sensitive property for the training data
       - if $o \in NN(q, k)$, then collision probability from H $\geq p_1$
       - if $o \notin NN(q, ck)$, then collision probability from H $\leq p_2$

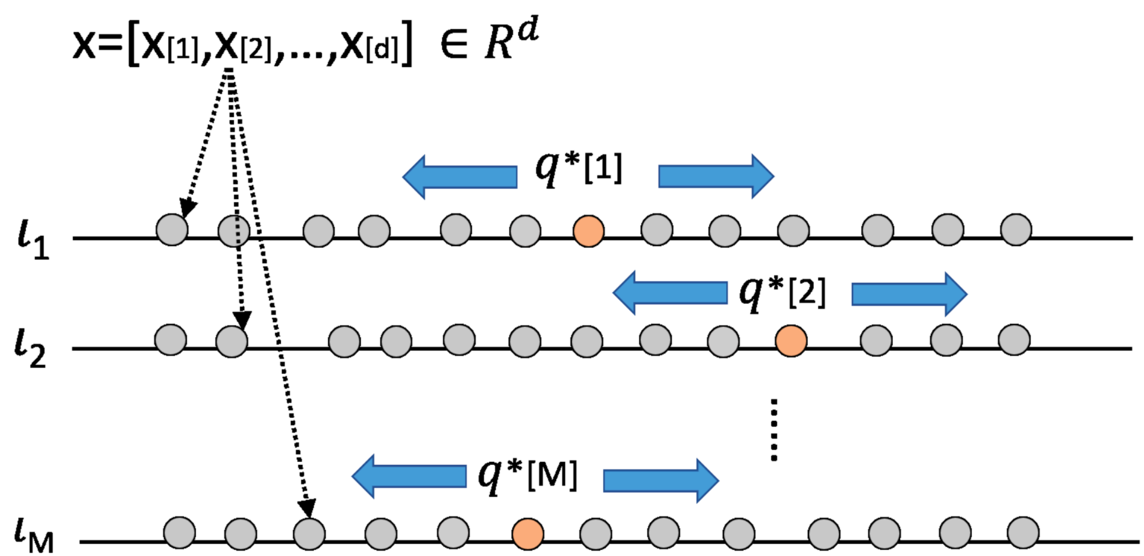
Key difference with AdaBoost: High recall and low precision

Learned ANN Index [LZSW+20]

- Focus on external I/O
  - Use B-trees and maximize the use of sequential I/Os
- Scheme:
  - H: $R^d \rightarrow R^M$
  - Index each dimension of H(X) in a clustered B-tree
- Query processing
  - Collect candidates on each of the M projected dimensions
  - When T candidates are seen on all M lists, rerank them and return top-k

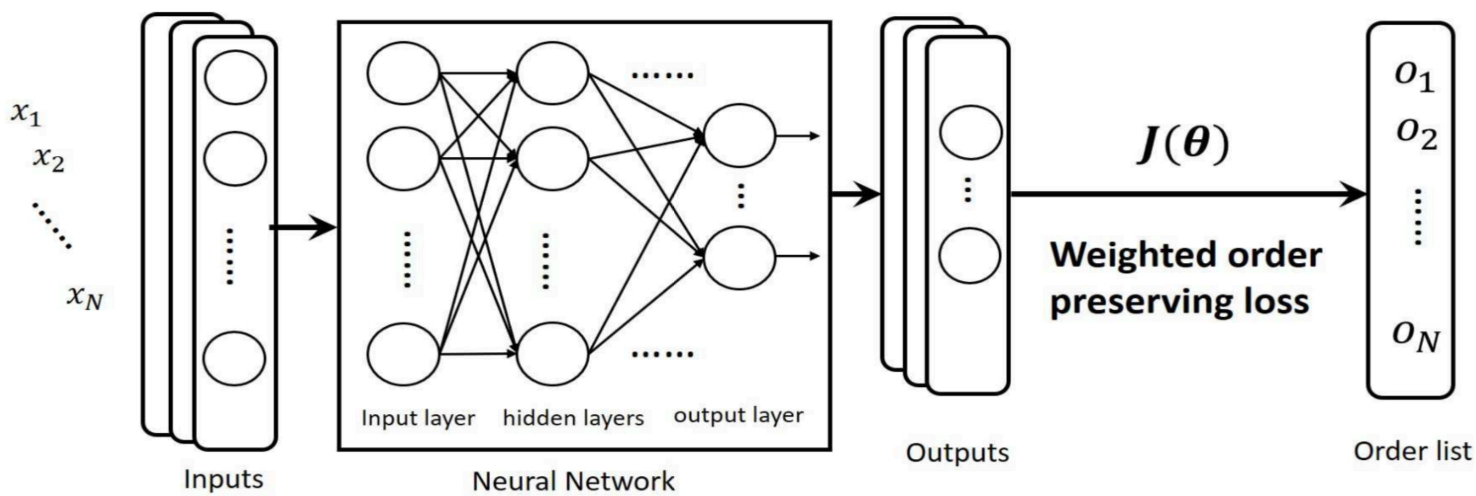
≈ MedRank (minfreq = 1) [FKS03]

must c.f., [AFKPS08]



Function family

- Consider
  - linear functions
    - $H(x)[m] = w_m^T x$
  - non-linear functions



W

How to learn the parameters?

Consider the linear functions:
$H(x)[m] = w_m^T x$

- Goal:
  - Encourage segment-order preserving mappings

Part of the Loss function

$$J^*(\mathbf{w}_m) = \sum_{i=1}^{L} \sum_{\tilde{x} \in l_i^o} \mathbf{1}_{r(\tilde{x}) \in [t\cdot(i-1), t\cdot i)}$$

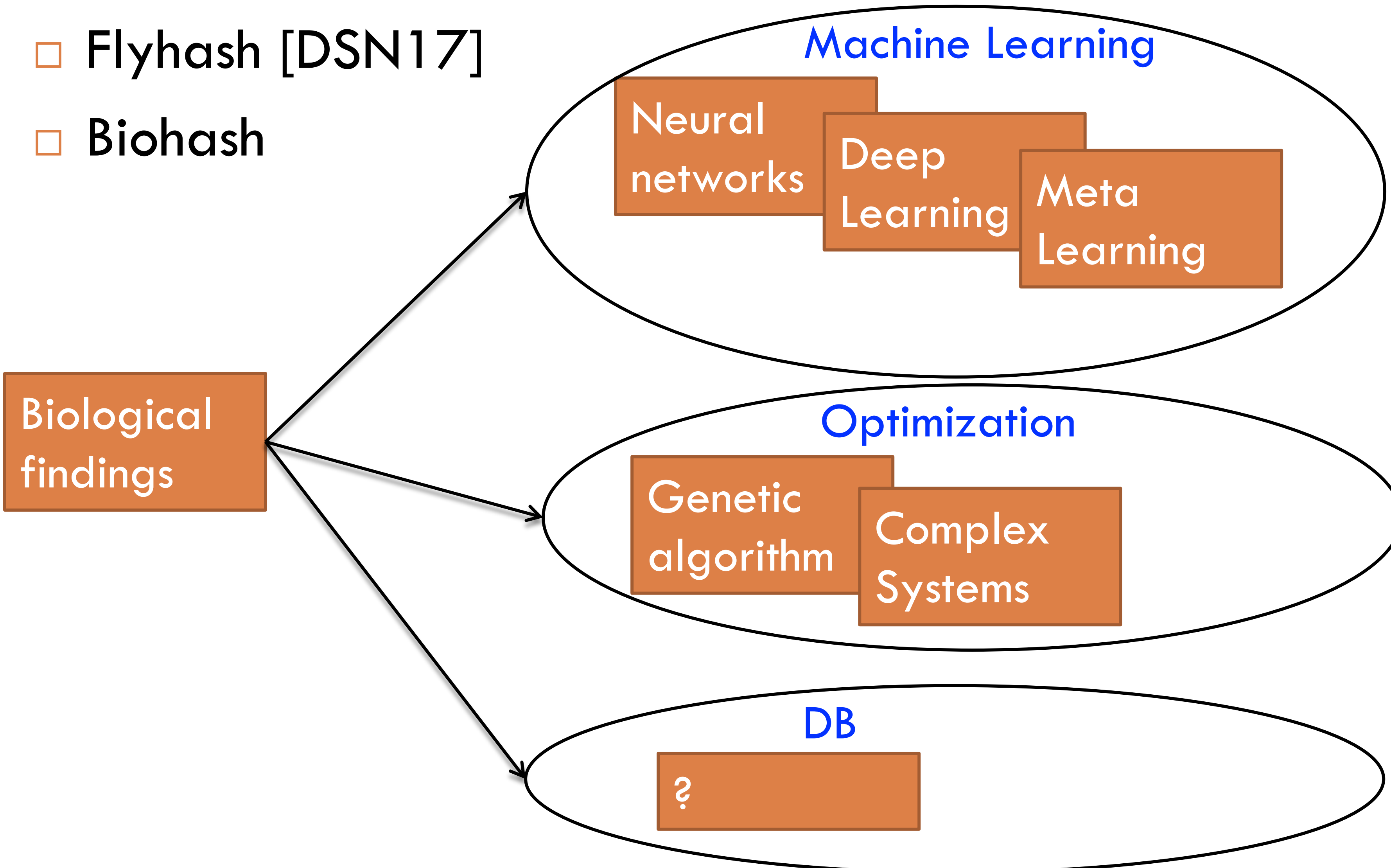
Continuous Relaxation

mapped x in the i-th segment

x in the i-th segment

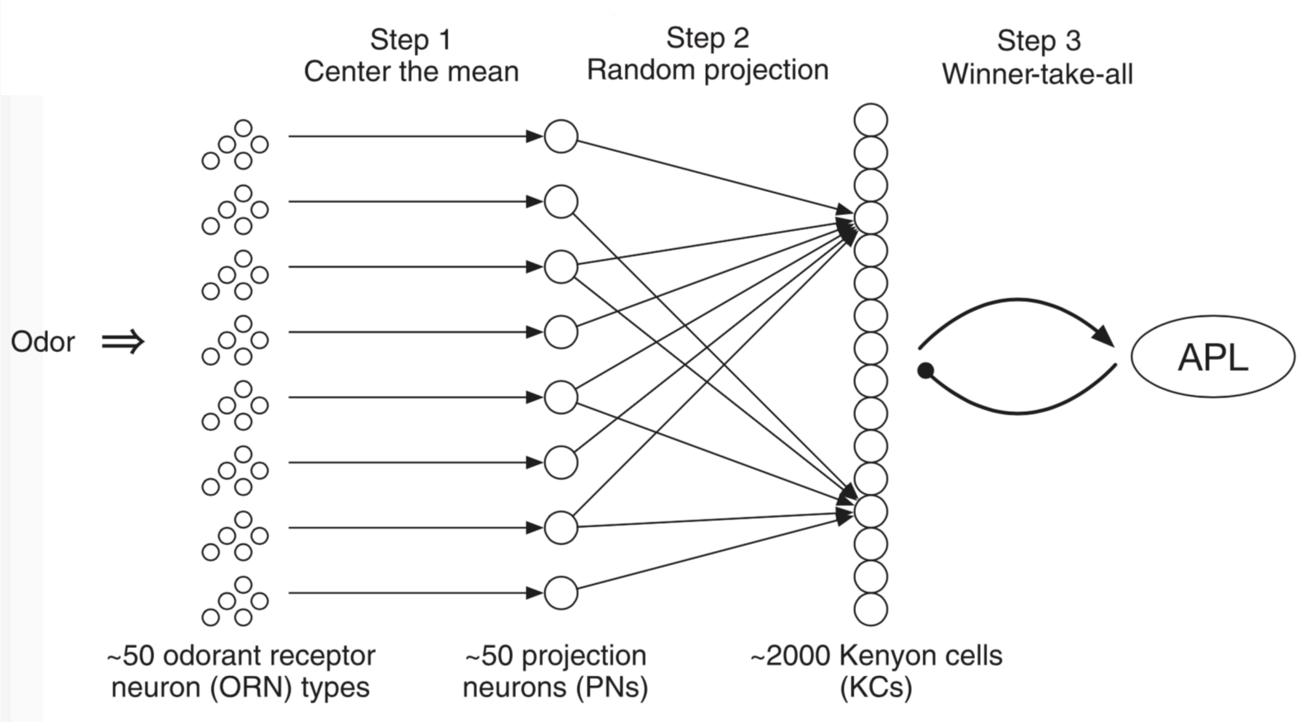
Biology Inspired Hashing

- Flyhash [DSN17]
- Biohash



FlyHash

- The fly olfactory circuit generates a low-overlapping, sparse neuron activation pattern when an odor is presented



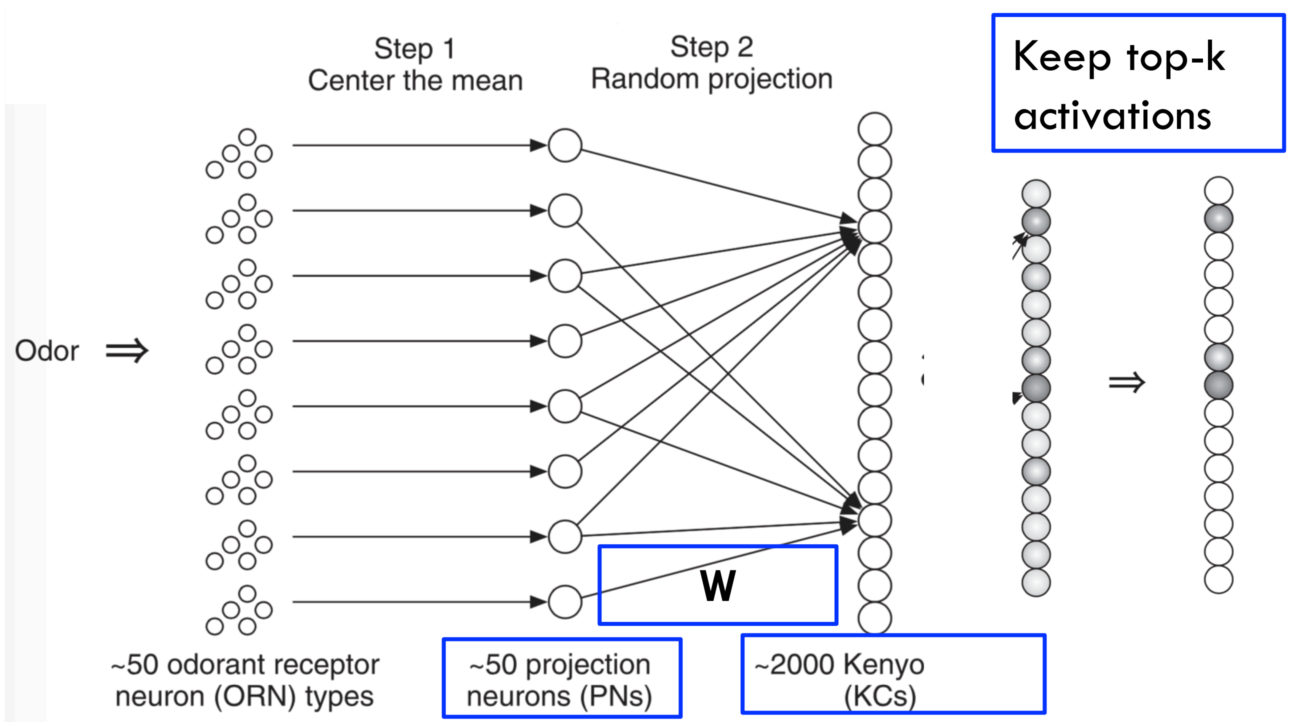
FlyHash

$$\mathbf{z} = \sigma(\mathbf{W}\mathbf{x})$$

- Difference with LSH
  - **W** is a sparse binary random matrix
  - Dimensionality expansion !!
  - Sparsification

Enable

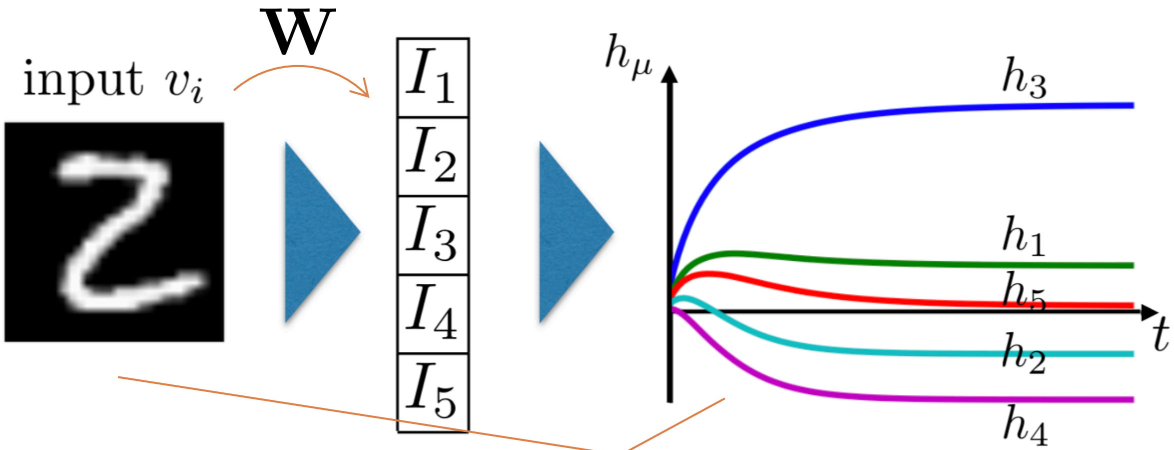
- L2 distance approximately preserved in expectation



[KH19]

- Unsupervised learning inspired by biological synaptic plasticity rules
- Overview
  - (Given W) Stabilizing the hidden competing neurons
  - Learning the projection matrix W

Learning h



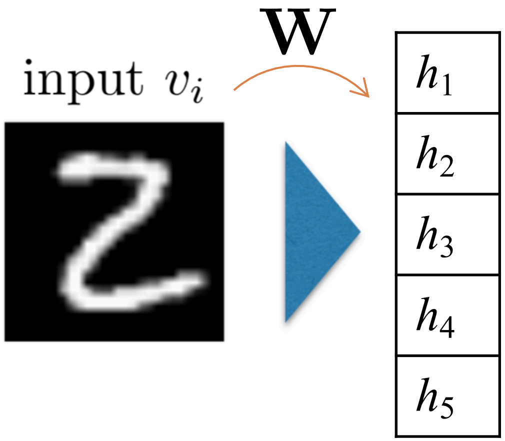
$$\tau \frac{\mathrm{d}}{\mathrm{d}\,t}\left(h_\mu\right) = I_\mu \boxed{- \gamma \sum_{\nu \neq \mu} \mathrm{RELU}\left(h_\nu\right)} - h_\mu, \quad \text{where} \quad I_\mu = \langle \mathbf{W}_\mu, \mathbf{v} \rangle$$

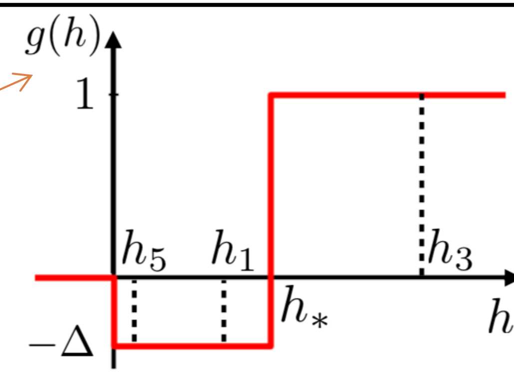
Competing for activation

- Fixing the W, the dynamical equation will converge to a stable hidden vector h

Learning W

h* is the activation threshold





$$\tau_L \frac{\mathrm{d}}{\mathrm{d}\,t}(\mathbf{W}_i) = g(Q)\left(\mathbf{v}_i - \langle \mathbf{W}, \mathbf{v} \rangle \mathbf{W}_i\right), \qquad \text{where } Q = \frac{\langle \mathbf{W}, \mathbf{v} \rangle}{\langle \mathbf{W}, \mathbf{W} \rangle^{\frac{1}{2}}}$$

p = 2, for any one hidden neuron
W is its corresponding weight vector
R = 1

Force |W|$_p$ to converge to R^p

- Fixing the h, the dynamical equation will converge to a final weight matrix W

BioHash – Learning W

input v_i

$$\xrightarrow{\mathbf{W}} \begin{array}{|c|}\hline h_1 \\ \hline h_2 \\ \hline h_3 \\ \hline h_4 \\ \hline h_5 \\ \hline \end{array}$$

sort the projections and rank them

$$g[z] = \begin{cases} 1, & \text{rank}(z) = 1 \\ -\Delta, & \text{rank}(z) = r \\ 0, & \text{otherwise} \end{cases}$$

$$\tau_L \frac{\mathrm{d}}{\mathrm{d}t}(\mathbf{W}_i) = g(\boxed{\{Q\}})(\mathbf{v}_i - \langle \mathbf{W}, \mathbf{v} \rangle \mathbf{W}_i), \quad \text{where } Q = \frac{\langle \mathbf{W}, \mathbf{v} \rangle}{\langle \mathbf{W}, \mathbf{W} \rangle^{\frac{1}{2}}}$$

sorting among different normalized inner products (each produced by a different $\mathbf{W}_i$)

- The rest is the same as FlyHash (i.e., WTA sparsification)