A Theoretical Analysis

Let $D = \{(x_i, y_i)\}_{i=1}^N$ be the set of (key, value) pairs, where x_i and y_i are i.i.d. samples of two independent distributions. Assume x_i's and y_i's are random variables uniformly chosen from $\{0,1\}^l$. Let W denote the vector of trained parameter of the neural network without the memory and $n = |W|$ the dimension of W. Note that after training W is a random variable that depends on x_i and y_i's.

Claim A.1. *Without neural memory problems 1.1, 1.2 with N key,value pairs from the above distribution cannot be learnt with accuracy better than $O(\sqrt{n/N})$*

Proof sketch. Let $g(x, w)$ be the function defined by the neural network, where x is the argument for the input, and w is the argument for the weight. Let $\hat{y}_i = g(x_i, W)$ be the prediction from the neural network. Simple information theoretical argument shows that $\frac{1}{N}\sum_{i=1}^N I(y_i, \hat{y}_i) \leq \frac{1}{N}\sum_{i=1}^N I(y_i; W) \leq \frac{H(W)}{N}$, where I is the mutual information and H is the entropy function.

Since the n weight parameters are represented using some finite bit precision, $H(W) \leq O(n)$ Therefore the average mutual information between prediction and true value is bounded by $O(n/N)$, from which one can show the accuracy is bounded by $O(\sqrt{n/N})$. See App. B for the full proof. □

Let E denote the $N \times d$ matrix denoting the outputs of the LSH layer for N keys. The entries of E are obtained from entries of memory contents $N \times s$ matrix Z by a linear transform involving the sketching matrices $R_1, .., R_k$ and the LSH hash indices. We will also assume that the keys are random and long enough so that the hash buckets are uniform random and independent (this is true in the hyperplane LSH for example if we use orthogonal hyperplanes). The following claim (proven in App. C) shows that any E can be obtained by inverting this linear transform to get a suitable Z and solving for it is a well conditioned problem (note that the condition number for matrix A is the ratio of its largest to the smallest singular value of A^tA).

Claim A.2. *Let R_B denote the linear transform that transforms entries in Z to entries in E (think of Z, E flattened into a single vector). With high probability, R_B^t has condition number at most $O(\log N)$ (under reasonable assumptions). Hence for any loss function L, $\nabla_Z L$ is 0 iff $\nabla_Z E$ is 0.*

Proof sketch. To get the intuition consider the case when $s = d = 1$; that is, the values stored in the buckets are one dimensional and the random matrices $R_1, .., R_k$ are 1×1 that are essentially scalars. In this case R_B^t can also be viewed as a bipartite graph with N nodes on the left (corresponding to keys) and M nodes on the right (corresponding to buckets) and Nk edges (corresponding to the hash lookups). The degree on the left nodes is k and by a balls and bins argument we can bound the maximum degree on the right by $O(\log N)$ with high probability (see for example Raab and Steger (1998)). Given this sparse random structure we can lower bound $|R_B^t x|_2^2/|x|_2^2$ by $k/2$ and upper bound it by $O(k \log N)$ giving a condition number bound of $O(\log N)$ □

Claim A.3. *With one layer of neural memory followed by a single linear layer denoted by matrix $A \in \mathbb{R}^{d \times l}$ problem 1.1, can be learnt using memory of size $M = \Omega(Nk \log N)$, $k = \Omega(\log N)$, $sk = \Omega(d)$ assuming A^t is well conditioned through out the gradient descent training. Further with gradient descent one can achieve an error below ϵ in $O(\kappa \log N \log(1/\epsilon))$ steps where κ is a bound on the condition number of A^t.*

Proof. Assume first for simplicity that there is no collision in the k LSH buckets for a certain key x_i from any other key and that A^t is fixed and well conditioned. Denote the corresponding value vector $y_i \in \mathbb{R}^d$. Then we are training the vectors of entries in the k the LSH buckets, denoted by $z_{h_1(x_i)}, ..., z_{h_k(x_i)}$, or $z_{h_1}, ..., z_{h_k}$ for brevity. The output is $\hat{y} = A(R_1 z_{h_1} + ... + R_k z_{h_k})$, where R_i are random sketching matrices. The loss is measured by

$$|y_i - A(R_1 z_{h_1} + ... + R_k z_{h_k})|_2^2,$$

where $|v|_2$ is the 2-norm of vector v. Let z_b denote a single vector obtained by concatenating $z_{h_1}, ..., z_{h_k}$ and R_b denote a single matrix obtained by stacking $R_1, .., R_k$ horizontally. Then $\hat{y} = R_b z_b$ and R_b^t is well conditioned as it is a sufficiently rectangular random matrix when $sk \geq \Omega(d)$ (see Rudelson and Vershynin (2009)). Since this loss function is strongly convex, in a gradient descent minimization the loss goes below ϵ in $O(\kappa \log(1/\epsilon))$ steps for that key,value pair, (Boyd and Vandenberghe, 2004).

Even if there may be collisions, we look at the matrix $\hat{Y}$ of all outputs by stacking together the outputs $\hat{y}$ for all keys. Let $\hat{y}_B$, e_B, z_B denote the flattened version of $\hat{Y}$, E, Z into single column vectors. Then$\hat{y}_B = A_B e_B = A_B R_B z_B$

where A_B is a block diagonal matrix with N copies of A along the diagonal and so A_B^t is well conditioned. From Lemma 3.2 the R_B^t has condition number at most $O(\log N)$ with high probability. Therefore $(A_B R_B)^t$ has bounded condition number. Even if A is allowed to be trained, the above argument holds as long as A^t is well conditioned throughout the gradient descent. □

Claim A.4. *The LSH hash function maps the fuzzy keys in a ball $B(x, \epsilon)$ into at most $N^{O(\epsilon)}$ hash buckets. Thus with neural memory, an instance of problem 1.2 with N fuzzy keys can be viewed as an instance of problem 1.1 with at most $O(N^{1+O(\epsilon)})$ (key, value) pairs.*

Proof. In the fuzzy (key, value) lookup problem instead of using a fixed key x, the query key is a random point r from a ball $B(x, \epsilon)$. The main idea is that even though the number of possible keys in $B(x, \epsilon)$ may be large, the number of hash buckets they get mapped to is bounded and at most $N^{O(\epsilon)}$ – this is proven by bounding the entropy of the distribution of the hash bucket-id $h(r)$ (for any one of the LSH hash functions h) given x to be at most $\log(N^{O(\epsilon)})$ based on the the ideas in (Panigrahy, 2006). Specifically we can obtain the bound for the entropy $I = H(h(r)|x) \leq O(\epsilon \log N)$. The number of buckets that cover a significant fraction of $B(x_i, \epsilon)$ is at most 2^I. For any one random hyperplane w Lemma 3 in (Panigrahy, 2006) shows that $H(\text{sgn}(r \cdot w)|x)) \leq O(\epsilon)$ which implies that from $\log m$ random hyperplanes $I = H(h(r)|x)) \leq O(\epsilon \log m) = O(\epsilon \log N)$. Lemma 2 in (Panigrahy, 2006) shows that $2^I = 2^{H(h(r)|x)} = N^{O(\epsilon)}$ buckets cover more than $1/I$ fraction of the ball. So by using $k > \tilde{O}(1/I) = O(1/\epsilon)$ LSH functions, with high probability most of each of the balls are covered. See the proof of Theorem 4 in in (Panigrahy, 2006) for the details. Therefore, the problem of N fuzzy (key, value) pairs essentially breaks down to a problem of $N^{1+O(\epsilon)}$ (bucket-id, value) pairs via the LSH hash functions. □

Claim A.5. *Running gradient descent using an embedding table of size N and the rest of the network of size n is equivalent to running gradient descent with a neural memory of size $O(Nk)$ and the same network initialization in the following sense: there is a one to one correspondence between parameter values in the two cases and a critical point of the first case is also a critical point of the second case, and vice versa.*

Proof. N is the size of the vocabulary corresponding to the embedding table. We will argue that training with an embedding layer is equivalent to training with neural memory access at the first layer. For simplicity first assume all the Nk buckets for the N words are distinct. In this case, we can interpret the output value of the memory layer $\sum_{j=1}^{k} R_j z_{h_j}$ to the embedding entry for a lookup word, this is because in the back propagation, the gradient coming above the summation node can be viewed as the gradient coming to the embedding entry e_i in the case when there was an actual embedding layer. Even if there may be collisions, let E denote the $N \times k$ matrix of embeddings obtained for the N words based on the hash the lookups into Z. The transform of entries from Z to E is linear and that linear transform has condition number at most $O(\log N)$ with high probability (see Lemma 3.2) Therefore, $\nabla_Z L = 0$ iff $\nabla_E L = 0$.

□

A.1 LSH as Kernel

Consider one layer of the LSH memory that stores scalar values at each bucket and on an input simply outputs the sum of the values stored in the retrieved hash buckets. We will show that by storing a scalar value z_i in the ith hash bucket, one can view the LSH table as a kernel that projects x into a k-sparse M dimensional vector that indicates the buckets an input is mapped to. If we just take the sum of the z_i values stored in the retrieved buckets we get the function $f(x) = \sum_{j=1}^{k} z_{h_j(x)} = z \cdot \Phi(x)$, a dot product of vectors $z, \Phi(x)$ where z is the vector of values in the hash table and $\Phi_i(x)$ is the indicator bit that x get mapped to i under one of the k hash functions.

Let $p(x, x')$ denote the probability that x, x' get mapped to the same bucket under random W. If x, x' differ by angle θ then for the hyperplane LSH $p(x, x')$ is the probability that none of the $\log m$ hyperplanes separate the points (recall m is the size of each hash table), which is $\psi(\theta) = (1 - \theta/\pi)^{\log m} \approx m^{-\ln e/\pi.\theta}$. Note that $E[\Phi(x) \cdot \Phi(x')]/k = p(x, x')$

Thus this LSH kernel maps to the kernel $K(x, x') = (1 - \arccos(x.x')/\pi)^{\log m}$. Thus the kernel function has a Taylor expansion given by $(1 - \arccos(x)/\pi)^{\log m}$. Based on methods from (Arora et al., 2019; Du et al., 2019) this can be used to show that even just one layer of LSH memory with a linear output node can learn polynomials – it can learn $(\alpha.x)^p$ (α is some unit vector) with generalization error $O(\sqrt{mp^2/n})$ for large enough k, where n is

the number of training examples. Note that if the number of training examples $n \gg mp^2$ this is negligible (see App. A.1).

Claim A.6. *The LSH lookup can be viewed as a kernel transform with kernel function $K(x, x') = P(x \cdot x')$ where $P(t) = (1 - \arccos(t)/\pi)^{\log m}$ represents a kernel function. The coefficient of t^p is at least $\Omega(\frac{1}{mp^2})$*

Proof. First note

$$1 - \frac{\arccos(t)}{\pi} = 1 - \frac{\pi/2 - \arcsin t}{\pi} = \frac{1}{2} + \frac{\arcsin t}{\pi}.$$

So

$$\begin{aligned}\left(1 - \frac{\arccos(t)}{\pi}\right)^{\log m} &= \left[\frac{1}{2}\left(1 + \frac{2\arcsin t}{\pi}\right)\right]^{\log m} \\ &= \frac{1}{m}\left(1 + \frac{2\arcsin t}{\pi}\right)^{\log m}.\end{aligned}$$

Note $\arcsin t$ has a Taylor series $t + \frac{1}{2}\frac{t^3}{3} + \frac{1\cdot 3}{2\cdot 4}\frac{t^5}{5} + \frac{1\cdot 3\cdot 5}{2\cdot 4\cdot 6}\frac{t^7}{7} + ...$ This has a radius of convergence of 1, and the coefficient of t^k is at least $1/k^2$ for odd k. So for $(1 + 2\arcsin t/\pi))^{\log m}$ just from the constant term the same lower bound on coefficient of odd powers follows. For even powers we can use the linear term from one of the factors. □

As in Arora et al. (2019); Du et al. (2019) we will argue that since a single LSH hash function can be viewed as a kernel function, if we have a sufficiently large number of such units k we learn any polynomial function. We will show how to obtain an output y_i for input x_i that is some scalar function of x_i; y is the vector of all y_i's. Let n denote the number of data points.

Since on an input, certain k buckets out of the mk buckets are accessed, we can represent the set of buckets accessed as a binary mk dimensional vector $h(x)$ that is 1 wherever the bucket is accessed. Let w denote the vector of values stored in all the mk buckets. If we just view a single LSH layer, with a linear node at the top where we simply add up the values stored in the buckets that are accessed, then the output is $w^t h(x)$ Let h be a $n \times mk$ matrix denoting all the vectors $h(x)$ stacked together as rows for all n data points.

As in Arora et al. (2019); Du et al. (2019) we will argue that for large enough k for fixed randomly chosen LSH function, if we train the values w stored in the buckets, the generalization error is at most $\sqrt{\frac{y^t (H^\infty)^{-1} y}{n}}$ where $H^\infty = E[hh^t]$. Note that as $k \to \infty$, hh^t concentrates to its expected value.

To see this more generally first look at the case when x is one dimensional to get the main point: since the Taylor series corresponding to $K(x, x')$ consists of all degrees of (xx') we can view it as taking dot product in the Kernel space that maps x to $\Phi(x) = (1, x, x^2, ...,)$ which is an infinite dimensional Hilbert space. $K(x, y) = \Phi(x).\Phi(y)$ (where the dot product operator may weigh different coordinates differently). Now we will argue that for any distinct set of inputs $x_1, .., x_n$ the set of vectors $\Phi(x_i)$ are linearly independent – it's suffices to prove linear independence even if we just focus on the first n columns which produces the Vandermonde matrix obtained by stacking rows $1, x, x, ..., x^{n-1}$ for n different values of x – this Vandermonde matrix cannot be singular as otherwise it would mean that a degree $n - 1$ polynomial (corresponding to the linear combination of columns that would give 0) can have n distinct roots $x_1, .., x_n$ which is impossible. Even if x is multidimensional and all the values along some dimension are distinct the same reasoning holds as we can ignore the other dimensions in $\Phi(x)$. The same reasoning applies even if x is multidimensional as long as their projection along some direction is distinct for all the n points as we can rotate the coordinate system accordingly. Now if all the points are distinct and far apart from each other, probability that a given pair coincides under a random projection is negligible and from a union bound the probability that any pair coincide is also bounded – so there must be directions so that projections along that direction are all distinct. For large enough k, $(hh^t)^{-1}$ approaches $(H^\infty)^{-1}$

If we are only training the vector w then it becomes a linear problem and given by a w s.t. $hw = y$. Let us assume that k is large enough so that hh^t is close enough to H^∞ and is invertible.

Lemma A.7. *If we train the weights w stored in the LSH buckets, then there exists a solution w with norm $y^t (hh^t)^{-1} y$*

Proof. The minimum norm solution to this is given by $w = (h^t h)^{-1} h^t y$ The norm $w^t w$ of this solution is given by $y^t h (h^t h)^{-2} h^t y$

We claim that $h(h^t h)^{-2} h^t = (hh^t)^{-1}$ To show this, we use the SVD. Let $h = USV^t h(h^t h)^{-2} h^t = USV^t (VS^2 V^t)^{-2} VSU^t = USV^t VS^{-4} V^t VSU^t = US^{-2} U^t = (hh^t)^{-1}$

Therefore, the norm of the minimum norm solution is $y^t (hh^t)^{-1} y$. □

For large enough k, $(hh^t)^{-1}$ approaches $(H^\infty)^{-1}$ – if we assume that the lowest eigenvalue $\lambda_{min}(H^\infty) \geq \lambda_0$ then the difference becomes negligible with high probability for large $k = poly(n/\lambda_0)$. As in proof of Theorem 6.1 in Arora et al. (2019) if $y = (\alpha.x)^p$ (α is some unit vector), $y^t H^\infty y = O(mp^2)$ as the coefficient of $(x.y)^p$ in the Taylor series is $1/O(mp^2)$. Based on this the Rademacher complexity of this function class is at most $\sqrt{\frac{y^t (H^\infty)^{-1} y}{n}}$. which is at most $\sqrt{mp^2/n}$

B Proof of claim 3.1

We give the full proof to claim 3.1 in this section.

Proof. It is sufficient to assume that the values are just one dimensional random bits. Let $g(x, w)$ be the function defined by the neural network, where x is the argument for the input, and w is the argument for the weight. Let $\hat{y}_i = g(x_i, W)$ be the prediction from the neural network. We have

$$I(y_i, \hat{y}_i) = I(y_i, g(x_i, W)) \leq I(y_i, (x_i, W)) = I(y_i, W) \tag{1}$$

where the first inequality follows from data processing inequality and the last equality is due to the independence of x_i and y_i.

Now consider

$$\begin{aligned} I(y_1, y_2, ..., y_N; W) &= \sum_{i=1}^{N} I(y_i; W | y_{i-1}, y_{i-2}, ..., y_1) \\ &\geq \sum_{i=1}^{N} I(y_i; W). \end{aligned}$$

By definition

$$I(y_1, .., y_N; W) \leq H(W).$$

Suppose the weight of the neural network is saved in a data structure of b bits (say, $b = 32$), then the total number of possible W's is 2^{bn}, and hence $H(W) \leq log_2(2^{bn}) = bn$.

Therefore we get

$$I(y_1, ..., y_N; W) \leq bn \Rightarrow \sum_{i=1}^{N} I(y_i; W) \leq bn \tag{2}$$

Combine (1) and (2), we get

$$\frac{1}{N} \sum_{i=1}^{N} I(y_i, \hat{y}_i) \leq \frac{1}{N} \sum_{i=1}^{N} I(y_i; W) \leq \frac{bn}{N}. \tag{3}$$

Suppose $P(\hat{y}_i = y_i) = p = \frac{1}{2} + \frac{1}{2}q$, then

$$\begin{aligned} I(y_i, \hat{y}_i) &= H(y_i) - H(y_i|\hat{y}_i) = H(1/2) - H(p) \\ &= 1 + p\log p + (1-p)\log(1-p) \\ &= (1/2)((1+q)\log(1+q) + (1-q)\log(1-q)) \\ &= \frac{3}{2\ln(2)}q^2 + \frac{7}{12\ln(2)}q^4 + ... \\ &\geq \frac{3}{2\ln(2)}q^2. \end{aligned}$$

Plug into (3), we get

$$\frac{1}{N}\sum_i q_i^2 \leq \frac{2b\ln(2)}{3}\frac{n}{N} \Rightarrow \frac{1}{N}\sum_i |q_i| \leq \sqrt{\frac{2b\ln(2)}{3}}\sqrt{\frac{n}{N}},$$

by convexity of x^2 function. □

C Condition number of R_B

Let Z be a $M \times s$ matrix denoting the contents of the LSH memory buckets. Let E denote the $N \times d$ matrix denoting the outputs of the LSH layer for N keys. The entries of E are obtained from entries of Z by a linear transform involving the sketching matrices $R_1, .., R_k$. Let e_B, z_B denote flattened versions of E, Z where the rows are concatenated to get a single column vector. Then $e_B = R_B z_B$ where R_B can be viewed as block sparse with $N \times M$ blocks of size $d \times s$ each – for each key x_i corresponding to the jth hash function h_j there is a block with value R_j at position $h_j(x_i), i$. We will also assume that the keys are random and long enough that all the hash buckets will be random and independent (this is true in the hyperplane LSH for example if we use orthogonal hyperplanes). We will now show that the transform R_B is well conditioned by bounding the condition number of R_B^t.

Lemma C.1. *With high probability, R_B^t has condition number at most $O(\log N)$ as long as $M = \Omega(Nk\log N)$, $k \geq \Omega(\log N)$, and $sk \geq \Omega(d)$. For any loss function L, $\nabla_Z L$ is 0 iff $\nabla_Z E$ is 0.*

Proof. To simplify the proof we will assume that $s = d = 1$ but the same proof holds for larger s, d. In this case the random matrices $R_1, .., R_k$ are 1x1 matrices and can be represented by scalars $r_1, .., r_k$ R_B is a $M \times N$ matrix that represents how for a key x_i, the contents of the M are accessed and linearly combined to obtain a single value for that key. R_B is sparse and for every key x_i has value r_j at position $h_j(x_i), i$. $R_B z$ is the vector of values for each key. output by the LSH layer.

We will upper and lower bound $|R_B^t x|_2/|x|_2$. R_B^t can also be viewed as a bipartite graph with N nodes on the the left (corresponding to keys) and M nodes on the right (corresponding to buckets) and Nk edges (corresponding to the hash lookups) – from vector x, x_i is supplied into the ith nodes on the left and the values are propagated along the edges (weighted by edge weight) and accumulated into the buckets giving the vector $R^t x$. The probability that $h_j(x_i)$ collides with another hash function another key is at most kN/M and the expected number out of the k hash buckets for a given key that see any collisions is k^2N/M. If $M = \Omega(Nk\log M)$ and $k = \Omega(\log N)$, with high probability at least $k/2$ hash buckets for each key are distinct and unique. Since r_i are ± 1, $R_B^t x$ will have at least $k/2$ entries with magnitude x_i corresponding to these unique buckets. So $|R_B^t x|_2^2 \geq (k/2)|x|_2^2$ (we note that eigenvalues of sparse random graphs have also been studied in Tran et al. (2013); Krivelevich and Sudakov (2003)

To upper bound $|R_B^t x|_2^2$ we make use of the fact that the maximum number of colliding keys in a single bucket is at most $O(\log Nk) = O(\log N)$ w.h.p – this follows from the balls and bins literature that analyses maximum load when Nk balls are randomly thrown into $M \geq Nk$ bins (see for example Raab and Steger (1998)). So any row of R_B has at most $t = O(\log N)$ non-zero entries. Now from the convexity of the $f(x) = x^2$ function, for any row r of R, $(rx)^2 \leq t\sum(r_i x_i)^2 \leq O(\log N)\sum_{r_i \neq 0} x_i^2$. So $|Rx|_2^2 \leq O(\log N)\sum_{i,j:R_{i,j}\neq 0} x_i^2 = O(\log N)k|x|_2^2$.

If s, d are more than 1 the same reasoning holds at a block level. Note that if $sk \geq \Omega(d)$ then x_i is a block on a node on the left side of the bipartite graph and sends blocks $R_1^t x_i, ., R_k^t z_k$ to different hash buckets – and the total norm of these blocks $|R_1^t x_i|_2^2 + .. + |R_k^t x_i|_2^2$ depends on the norm x_i and the condition number of the matrix

$[R_1^t, R_2^t, .., R_k^2]$ obtained by stacking the individual matrices vertically – if $sk = \Omega(d)$ this stacked matrix is well conditioned. The rest of the argument is same as before except that we replace scalar entries in x to a block of entries in the vector x.

So from back propagation chain rule $\nabla_{z_B} L = R_B^t \nabla_{e_B} L$. So $\nabla_Z L$ is 0 iff $\nabla_E L$ is 0. □

D Experiment Implementation Details

In this section, we provide implementation details. We use Tensorflow/Keras as the deep learning framework. We use random separating hyper-planes as the LSH function, with random hyper-planes chosen from standard normal distribution. In all experiments, for simplicity we chose the width of the fully connected dense layer to be equal to the sketch dimension (which is set to be 50 for most experiments, unless otherwise mentioned). The modification from standard neural networks to incorporate external memory is mostly minimal. As an example, the network used for tasks in Section 4.1 is given in Listing 1. Basically the output from each layer is augmented with retrievals from the memory using the output as the key, before feeding into the next layer. Finally, for all experiments to train the network, we use Yogi optimizer (Zaheer et al., 2018) with a learning rate of 0.001. We use $\beta_1 = 0.9$ for dense parts of the network, but $\beta_1 = 0$ for the memory block. This makes the gradient update on the learnable part of memory to be sparse, i.e. only occurs for indices accessed in the current example. Because if $\beta_1 > 0$, we might have non-zero momentum even for indices which are not used in the current training example and would need updating. But updating all of the memory block would slow down training as the memory size is large. Our choice of $\beta_1 = 0$ makes the update to be constant time. (Note the second order statistics in Yogi optimizer has sparse updates even for $\beta_2 > 0$, unlike Adam. So we can keep $\beta_2 > 0$ for the memory block.)

```
class SketchMem(tf.keras.Model):
  def __init__(self, hidden_size, sketch_dim, num_layers=1):
    super(MemNet, self).__init__()
    self.hidden_size = hidden_size
    self.sketch_dim = sketch_dim
    self.layer_list = [tf.keras.layers.Dense(hidden_size) for _ in range(num_layers)]
    self.memory = SimpleLSHMemory(hidden_size=hidden_size, log_num_buckets=20,
   num_hash_fn=5)

  def call(self, x):
    for l in self.layer_list:
      x = self.memory(x)
      x = tf.nn.elu(l(x))
    return x
```

Listing 1: Memory network implementation

E Using min-hashes to get a Knowledge graph

So far we assumed that the memory table is indexed by a dense vector. However the key could also be a set (of objects or sketches). In this case it makes more sense to use min-hash to obtain the bucket id. Further in each bucket we could store a list of top k most frequent buckets that are accessed with this bucket; thus the similarity hash table with list now becomes an adjacency list graph which can essentially be viewed as a knowledge-graph. Now lets say a certain pair of objects (A,B) have been frequently seen together. Now when the object A arises it would make sense to look up the sketch of the pair (A,B) that would be stored in the bucket corresponding to A and B and vice versa.