Sketch based Memory for Neural Networks

Rina Panigrahy
Google Research

Xin Wang
Google Research

Manzil Zaheer
Google Research

Abstract

Deep learning has shown tremendous success on a variety of problems. However, unlike traditional computational paradigm, most neural networks do not have access to a memory, which might be hampering its ability to scale to large data structures such as graphs, lookup-tables, databases. We propose *a theoretical framework for a neural architecture* where sketch based memory is integrated into a neural network in a uniform manner at every layer. This architecture supplements a neural layer by information accessed from the memory before feeding it to the next layer, thereby significantly expanding the capacity of the network to solve larger problem instances. We show theoretically that problems involving key-value lookup that are traditionally stored in standard databases can now be solved using neural networks augmented by our memory architecture. We also show that our memory layer can be viewed as a kernel function. We show benefits on diverse problems such as long tail image classification, language model, large graph multi hop traversal, etc. arguing that they are all build upon the classical key-value lookup problem (or the variant where the keys may be fuzzy).

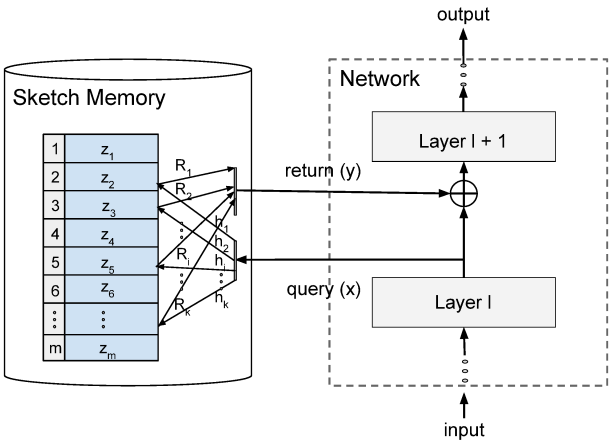
1 Introduction

Memory is an integral part of human learning and plays an important role in all of our daily activities and decision making. For example, if past events could not be remembered, it would be impossible for language, relationships, or personal identity to develop (Eysenck, 2012). Thus, it might not be unreasonable to assume artificial intelligence would also require similar capabilities of memorization. Traditional von Neumann computation paradigm captures such notions of memory: working memory (RAM) and episodic memory (hard disk); this enables it to precisely store and process vast amounts of information.

Neural computation paradigm, on the other hand, is focused on generalization by "memorizing" the function (approximately) in a set of trained edge weights, which has led to huge successes (Devlin et al., 2018; He et al., 2016b; Oord et al., 2016). Notably, product key memory (Lample et al., 2019) enhances this type of long term memory without much increase in computation. Specifically, it converts the usual weight matrix by input vector calculation in a dense layer to a k-nearest neighbor look-up of the input vector across a much larger matrix, which can be considered as the memory. Furthermore, some recent work, like DNC (Graves et al., 2016, 2014; Grefenstette et al., 2015; Kurach et al., 2016; Kaiser and Sutskever, 2015), also have tried to augment neural networks with different types of scratchpad as working memory in a problem specific manner. In general, however, there is not a unified way to equip neural networks with memory nor do we have a good theoretical understanding whether memory brings anything fundamental to the table.

In this paper, we propose a **theoretical framework SketchMem**, of an external memory architecture for long term episodic memory. The idea of sketching to store a complex object has been proposed in Ghazi et al. (2019); Panigrahy (2019); the main idea is to computing a a recursive sketch that capture the essential properties of a complex input as it is processed by a modular deep network – such recursive sketches can be used to compare so that underlying by just looking at the similarity between two sketches. Our framework can be viewed as a simplification of this sketching method and an abstraction of other mentioned proposals for memory for specific applications such as BERT (Lample et al., 2019). The main intuition behind our theoretical framework stems from the key advantage of having explicit episodic memories provides better **scalability** to larger problem instances. Certainly a deep network of a fixed size cannot solve problems that need more information than can be retained by all the parameters in the network. For example, consider a

Proceedings of the 24[th] International Conference on Artificial Intelligence and Statistics (AISTATS) 2021, San Diego, California, USA. PMLR: Volume 130.



Algorithm 1 SketchMem

Input: $x \in \mathbb{R}^d$
Parameters: Random matrices $R_i \in \mathbb{R}^{d \times s}$, independent hash functions $\text{LSH}_i : \mathbb{R}^d \to [M]$ for $i = 1, 2, ..., k$ and a memory table Z
for $i = 1 \to k$ **do**
  Hash input: $h_i = \text{LSH}_i(x)$
  Retrieve from memory: $z_{h_i(x)}$
end for
Compute sketch: $y = \sum_i R_i z_{h_i(x)}$
Return y

Figure 1: Network augmented with memory. **Left:** Schematic showing the overview of our sketch memory and the new data flow at each layer l to layer $l+1$, which is now augmented by adding data from the external memory. **Right:** Algorithm for retrieving relevant entries from the global memory, which would be added to normal data.

classification problem involving a large product catalog; as the number of products grows it may be harder to fit the information in the network weights. Providing a database or key-value lookup table would benefit the task. We formalize this insight of requiring a key-value lookup as two canonical problems which we believe are at the heart of most tasks that make use of memory.

Problem 1.1 (Key-value lookups)**.** *This is simply the problem involving a large number N of key, value pairs (x_i, y_i) where on query $f(x_i) = y_i$. Clearly if $N \gg n$ (n is the size of the network) this is impossible to learn (Claim 3.1). We prove that this can be learnt using our architecture (Claim 3.3)*

Problem 1.2 (Fuzzy key-value lookups)**.** *An extension of the previous problem where the input may be a point in a small ball around true center x_i. So $f(x_i + \vec{\epsilon}) = y_i$ where $\vec{\epsilon}$ is a vector of small length ϵ (Assuming x_i's are far enough that balls $B(x_i, \epsilon)$ of radius ϵ centered at x_i are disjoint. Again we prove that this function can be learnt by our architecture (Claim 3.4)*

Problem 1.2 takes our notion of key-value lookup to real-world problems, where we often do not have access to the exact key, but some noisy version of it. For example we may only see images of different products and we need to remember the name of each product or for spam classification, we have spelling or URL variations. Feldman (2019) studied a special case of Problem 1.2 and showed that memorization is necessary for good performance on such problems, thus reinforcing our claim that efficient external memory can be useful for such memorization. Here, (as in Feldman (2019)) we would like to point out that we don't look for the memory enhanced network to generalize to new key-value pairs, au contraire, we want it to stick to facts. We view each key-value pair as a distinct fact that cannot be inferred from other pairs. Generalization is limited to the encoding and retrieval step around the fuzziness of the key. In other words, our focus lies in showing external memory can help neural networks efficiently encode, store, and retrieve the known key-value pairs.

Based on the conceptual understanding from studying problems 1.1 and 1.2, ss part of SketchMem, we propose a simple neural memory module that plugs into most existing neural networks. Internally it uses a LSH (Locality Sensitive Hashing) table as the data structure for the episodic memory. While other hash functions such as product-quantization (Jegou et al., 2010; Lample et al., 2019) can be used, we work with LSH here due to its simplicity and ease with which it can be analyzed theoretically. We recommend to interface SketchMem at every layer of the deep network. It works as follows: At each layer, we hash the input into a set of hash buckets, the contents of which are sketched into a single vector that is added to the input into the next layer (see Figure 1 for a schematic and details are in Sec. 2).

We will theoretically show the expressive power of SketchMem (Sec. 3). We show in Claim 3.1 that for a network of size n without neural memory problems 1.1, 1.2 with $N \gg n$ (key,value) pairs cannot be learned better than a tiny accuracy – no better than $O(\sqrt{n/N})$. Whereas, with neural memory of size $O(N \log N)$ problem 1.1 can be learned perfectly (Claim 3.3), and 1.2 can be learnt within error at most ϵ using memory of size $O(N^{1+O(\epsilon)} \log N)$ (Claim 3.4). We also show that the external memory can be used to simulate a large embedding table – any problem that can be learned using an embedding table of size N and network of size n (excluding the embedding table) can be learnt via neural memory of size $O(N \log N)$ and a network of size n (Claim 3.5). We also show that our LSH based memory layer acts like a kernel transform; a single layer of the LSH based memory access can be viewed as a kernel transform with kernel func-

tion $K(x, y) = (1 - \arccos(x.y)/\pi)^c$ (where c is some parameter) (Claim 3.6).

We also experimentally demonstrate (Sec. 4) the scalability and efficiency of SketchMem. In particular, we gain improvements on long-tail image classification and entity resolution tasks. Finally, in Sec. 5, we conclude by discussing the present work in context of other memory augmented neural networks.

2 Architecture

The main idea is to use similarity preserving hashing to store information associated with each layer output in the neural-memory, and retrieve them during inference when similar layer outputs are produced in future; this is then added to the input to the next layer. For concreteness think of a network that takes as input a facial image and outputs at a certain layer an embedding of the facial features x that can be used to identify the person. Even for the same person the output of that layer may differ each time – so it makes sense to use a similarity preserving hash function to lookup (one or more locations) the neural memory. The contents of the neural memory are trainable and may be used to store useful metadata such as address, age, gender (though not expected to be in an interpretable format) for each person. If the number of persons is large we will clearly need an external neural memory as the information required to store all the metadata may be much larger than the capacity in the edges of the network.

Related objects from the neural memory are retrieved by creating (one or more) similarity preserving hashes of the output from a layer that is used to index into the neural memory. The similarity preserving hash can be implemented using similarity preserving sketching methods such as LSH (Locality Sensitive hashing, as described below). Let x denote the output of a layer in a neural network. The neural memory takes x as input query vector, and hashes x into k bucket ids using k independent LSH functions $h_1, ..., h_k$. Each bucket is addressed by its bucket id and contains a learnable vector. The tuple of learnable vectors $z_{h_1(x)}, ..., z_{h_k(x)}$ (for brevity, we denote these vectors $z_{h_1}, ..., z_{h_k}$) can be tuple-sketched into a single sketch vector by first padding each of them with zeros to make them all of the same dimension and then producing the sketch $y = R_1 z_{h_1} + .. + R_k z_{h_k}$ by using simple random subspace embedding matrices $R_1, ..., R_k$. See Figure 1 for a schematic of the neural memory architecture. This sketch vector y is then added to the layer output x to be fed to the next layer (after possibly dropping sufficient number of trailing dimensions from y to match the dimension of x).

Locality Sensitive Hashing (LSH) is a popular variant of hashing that tends to hash similar objects to the same buckets. Let us look at an LSH that maps an input to one (or a few locations) out of the m hash buckets. It is well-known that LSH provably provides sub-linear query time and sub-quadratic space complexity for approximate nearest neighbor search. More specifically, fix $0 < r_1 < r_2$, where r_1 is the threshold for nearby points, and r_2 is the threshold for far-away points, i.e. for $x, y \in \mathbb{R}^d$, we say x and y are nearby if $|x - y|_2 \leq r_1$ and they are far-away if $|x - y|_2 \geq r_2$, where $|x|_2$ is the 2-norm of the vector x. Let $c = r_2/r_1 > 0$ denote the distance gap as a ratio. Let $p_1 \leq Pr(h(x) = h(y) : |x - y|_2 \leq r_1)$ and $p_2 \geq Pr(h(x) = h(y) : |x - y|_2 \geq r_2)$ denote lower and upper bounds on the collision probability of nearby points and far-away points, respectively. Define $\rho = \frac{\log(1/p_1)}{\log(1/p_2)}$. Then LSH-based nearest neighbor search has a $O(n^\rho)$ query time and $O(n^{1+\rho})$ space complexity for a c approximate nearest neighbor query (Andoni et al., 2014; Andoni and Razenshteyn, 2015; Andoni et al., 2015).

In this work, we use random hyperplane based LSH (Charikar, 2002, described in the box) to sketch a vector into a hash bucket due to its simplicity, although other types of hashing such could be used as well – for example min-hash (Broder, 1997) could be used on a set or a tuple object to map that object to a discrete hash bucket. See the next paragraph for a brief description of the random hyperplane based LSH.

We first fix the notation for the rest of the paper. Let d be an upper bound on the width of each layer output; we assume that if layer outputs are of different lengths then we zero pad them to make them of width d. Let N be the number of (key, value) pairs, k be the number of LSH hashes, m be the number of buckets in one hash table, M be the total number of buckets in neural memory (note $M = mk$), $h_1(x), ..., h_k(x)$ be the indices of the k hash buckets for vector x, s be the dimensionality of the trainable vector in each bucket, r be the dimensionality of keys and values, and $Z \in \mathbb{R}^{M \times s}$ be the collection of learnable vectors of the LSH memory. The random hyperplane based LSH makes use of a random matrix $W \in \mathbb{R}^{d \times b}$, where $b = \log m$. For an input vector $x \in \mathbb{R}^d$, we first compute the vector $(\text{sgn}(x \cdot w_1), \text{sgn}(x \cdot w_2), ..., \text{sgn}(x \cdot w_b))$, where sgn() is the sign function and w_i is the i-th column of W. Then these b bits are concatenated together to get a $\log m$ bit index for a table of size m. To ensure that each hash function accesses a distinct section of the table of buckets, $h_i()$ accesses buckets starting at offset $m(i-1)$ so that the final index is in $[1..M]$. The

hyperplane hash function is chosen due to its simplicity and good performance in practice (Andoni et al., 2015).

3 Formal Guarantees for Learning Problems involving Memorization

We theoretically show the expressive power of SketchMem in this section.

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 and y_i are random variables uniformly chosen from $\{0,1\}^l$. Let W denote the trainable parameter of the neural network without the memory and $n = |W|$ the size of W. Note that after training W is a random variable that depends on x_i and y_i.

Claim 3.1. *Without SketchMem, instances of problems 1.1, 1.2 with N (key,value) pairs from the above distribution cannot be learnt with accuracy better than $O(\sqrt{n/N})$*

This follows from a simple information theoretic argument. See App. B for the full proof.

Next, we show instances of problem 1.1 can be solved by a SketchMem agumented network. Let E be 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 content matrix $Z \in \mathbb{R}^{N \times s}$ by a linear transform involving the sketching matrices $R_1, .., R_k$ and the LSH hash indices, as explained in Figure 1. 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, e.g. 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 condition number for matrix A is the ratio of its largest to smallest singular value of A^tA).

Claim 3.2. *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. For 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 left (corresponding to keys) and M nodes on 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 3.3. *With one layer of SketchMem 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 $M = O(Nk \log N)$, where $k = O(\log N)$ and $sk = \Omega(d)$, assuming A^t is well conditioned through out gradient descent training. Further, with gradient descent one can achieve error below ϵ in $O(\kappa \log N \log(1/\epsilon))$ steps where κ is a upper bound on 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 Claim 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 3.4. *The LSH hash function maps the fuzzy keys in a ball $B(x, \epsilon)$ into at most $N^{O(\epsilon)}$ hash buckets. Thus with SketchMem, 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 3.5. *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 SketchMem of size $O(Nk)$, $k = \Omega(\log N)$ 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 SketchMem 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 $\nabla_E L = 0$. □

We also show that our LSH based memory layer acts like a kernel transform. See App. A for more details.

Claim 3.6. *(informal) A single layer of the LSH based memory access can be viewed as a kernel transform with kernel function $K(x, y) = (1 - \arccos(x.y)/\pi)^{\log m}$.*

One can view the LSH table as a kernel that projects x into a k-sparse M dimensional vector $\Phi(x)$ that is a hot encoding of the buckets an input is mapped to. Since LSH tends to map similar inputs to similar set of buckets one can compute the expected value of the dot product $\Phi(x).\Phi(y)$ for two similar inputs x, y which turns out to be $K(x, y) = (1 - \arccos(x.y)/\pi)^{\log m}$. 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 polynomial functions.

4 Experiments

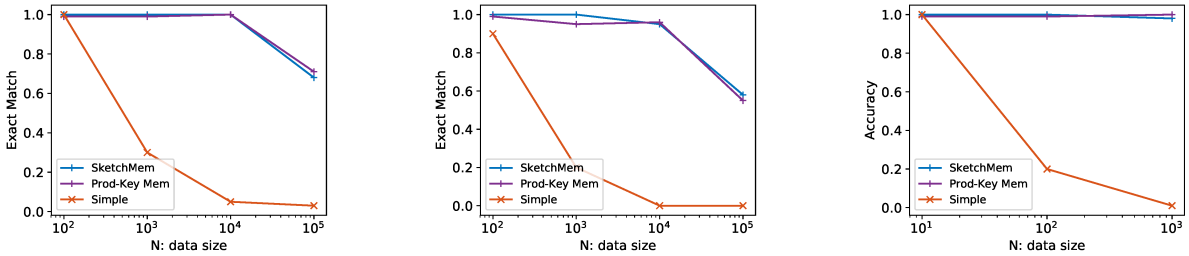
We now present empirical studies for our proposed architecture in order to establish that (i) SketchMem can scale to tasks involving memory; (Sec. 4.1), (ii) SketchMem is flexible and can be applied to different scenarios, e.g. in recurrent networks (Sec. 4.2), and (iii) SketchMem brings improvement in real world problems (Sec. 4.3). More experiment details are in App. D.

For all experiments we use SketchMem with $k = 5$ hash functions, total number of buckets $M = 5 \times 2^{20}$, dimensionality of the trainable vector in each bucket $s = 50$, width of a sketch vector $d = 50$, and memory is augmented to every layer, unless mentioned otherwise.

4.1 Warm-up

Crisp Key to Value Prediction We consider the key-value problem 1.1. It is set up as a regression from x_i to y_i with mean square loss. The keys are generated as random 50-bit vectors, i.e. $x_i \in \{0, 1\}^{50}$ and similarly $y_i \in \{0, 1\}^{50}$. We compare memory based SketchMem and Prod-Key Mem (Lample et al., 2019) against a simple neural network of depth 3 and width 50 with ELU non-linearity, totaling in $7,650$ parameters. In SketchMem, the neural network component is of the same size and structure, but memory lookups have been added after each layer. Prod-Key Mem has been chosen so that total size is similar to SketchMem. We see from Figure 2a that even with constant size neural network part, both SketchMem and Prod-Key Mem can leverage the external memory to correctly store and retrieve a large number of key-value pairs. Moreover, to achieve similar accuracy as SketchMem for $N = 10^4$ by simply increasing the neural network size, we would need internal width of 600, which would increase the computational cost to 18 million FLOPs compared to 400 thousand FLOPs for SketchMem with 50 width. Note that we are not looking to generalize to unseen key-value pairs, which is impossible. Instead we focus on known key-value relations and utilizing memory.

Fuzzy Key to Value Prediction We turn to the case of value retrieval when the key is noisy (problem 1.2). For this problem, we set up the data-set similar to the previous case with additional constraint that all keys are separated by $\sqrt{d}$ distance. Each time when



(a) Key-value problem (b) Fuzzy key-value problem (c) Set coherence problem

Figure 2: Performance of SketchMem on synthetic tasks as dataset size is increased. We compare it to a "simple" neural network which does not have access to external memory.

a key x_i is queried, we add a new random normal noise of variance $1/d$. Our network size and baseline is the same as the previous case. We want the neural network portion to only generalize for the noise, i.e. map fuzzy keys to correct locations in the memory. The results are reported in Figure 2b which indicate strong performance of SketchMem. Also the computational efficiency gains are similar to previous case.

Set Coherence Next we look at the fuzzy key-value problem where the keys are sets (such as a bag of words) and fuzziness is introduced by choosing random subsets. We choose sets of size 5 from a large universe and fuzzy versions are random subsets of size 3. Values are 50 dimensional random vectors as before. However instead of asking to produce the value for a given key, we give both the (fuzzy-set, value) pair and ask if that is the correct value for that fuzzy-set. In the YES case the fuzzy-set is truly a random subset of size 3 of the true set and in the NO case is some other random set of size 3. We observe (Figure 2c) that augmenting a fixed size neural network with the proposed memory architecture enables highly accurate predictions even with increasing N, the size of the underlying database. Alternative to using SketchMem, if we try to simply increase the network size to achieve similar error, the increase in computational cost is enormous as can be seen from Fig, 3.

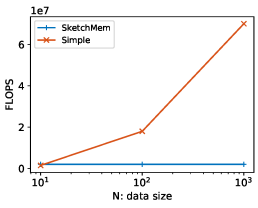


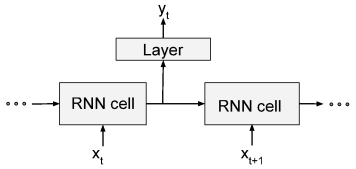
Figure 3: Baseline network computational cost with increasing data size N to maintain accuracy.

4.2 Simple RNN-based Applications

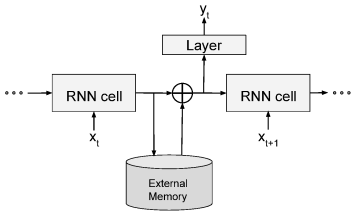
In this subsection, we explore the application of our neural memory to recurrent networks. In a standard RNN, the network operates on the input and previous state to produce the next state. We insert our interface to external memory along this temporal flow of states, which is a simple change from normal RNN as illustrated in Figure 4. In other words, we use "sketching" to store the state vector from each time-step in the neural-memory and pass on to the next time step the state by augmenting it with retrieved similar state sketches. We hope that this enables the network to quickly recall if in the past it encountered a similar situation and thus be better prepared to act.

Graph Traversal for Relation Predictions To exhibit the ability of memory augmented RNN to recall similar past events and act accordingly, we look at the task of 2-hop relation prediction, but with a knowledge graph hidden! Predicting 1-hop and 2-hop relations given the knowledge graph are standard problems in machine learning and have been part of TAC 2015 challenge (Ellis et al.). In our experiment, we assume there is a latent knowledge graph $\mathcal{G}$ with $\mathcal{E}$ as the set of entities and $\mathcal{R}$ as the set of relations. We do not show this graph and only provide training examples of the form: Input=(e_1, r_1), Output=e_2 corresponding to edge $(e_1, r_1, e_2) \in \mathcal{G}$ and Input=(e_1, r_1, r_2), Output=e_3 corresponding to a path (e_1, r_1, e_2), (e_2, r_2, e_3). Here $e_i \in \mathcal{E}$ and $r_i \in \mathcal{R}$. So in some sense we require the neural network to go over the training data, construct the knowledge graph internally, and then use it for relation prediction. For the experiment, we randomly generated graphs with $|\mathcal{R}| = 5$ and varied $|\mathcal{E}|$ from 10 to 5,000. All entities and relations are represented by 50-dimensional random binary vectors. We used a simple 50-dimensional RNN, but augmented with the proposed external memory. For each query, the state is initialized with the e_1 binary vector and at each time step the relation r_t binary vector is fed as input. The state of the RNN is used to predict the answer entity vector e_T for $T = 2, 3$. Again we use mean square error between predicted vector and true answer vector. As can be seen from Table 4c, without memory the RNN struggles to even predict 2-hop neighbors for very small graphs, whereas with memory we can scale till latent graphs with 5,000 nodes and 25,000 edges.

Language Model As another application, we chose to test our model on the tasks of word level language modeling. With a very simple 1 layer 50 dimensional LSTM (Hochreiter and Schmidhuber, 1997; Mikolov et al., 2010), we ran experiments on the Penn TreeBank



(a) Normal RNN



(b) SketchMem RNN

$\lvert\mathcal{E}\rvert$	Normal	SketchMem
10	1.00	1.00
100	0.27	1.00
1000	0.01	1.00
5000	0.03	0.89

(c) Accuracy on 2-hop relation prediction

Figure 4: Schematic for how normal RNN can be adapted to use SketchMem and result obtained using it on relation prediction task.

(PTB) dataset as pre-processed by Mikolov et al. (2010). PTB dataset has a vocabulary size of 10,000. We hope that having easy access to past occurrences of similar sentences should help in predicting the next word. The approach would be particularly helpful in predicting rare patterns, such as factual knowledge. We observe an improvement of more than 10 points in perplexity when the LSTM is augmented with external memory.

4.3 Real world applications

We now demonstrate capability of SketchMem in improving performance for real world problem which might involve handling large input and output spaces.

Spam Classification A natural problem with large input space is that of spam classification, for which we usually need to handle large vocabulary – e.g. there may be several spam links in emails and the number of these links may be much larger than the size of the network. A piece of text is considered to be spam if it has any tokens from a subset of the vocabulary which make the text to be classified as spam. We considered documents of length upto 20 tokens and represented them as a bag of words. In our experiment, we found a simple 3-layer SketchMem with specification as mentioned above to be able to perfectly classify spam text for total vocabulary size upto $30,000$ of which $15,000$ are considered to be spam tokens. Whereas the 3-layer network without memory struggled with even vocabulary of 1,000.

In some applications words maynot be inputs but maybe embeddings that come from some intermediate layer – for example the entire email text may come in as an image and the word embeddings may be output at some layer. We perform experiments where we give each word of the text as an image that goes through a set of CNN layers and outputs an embedding of the word – note that these embeddings may be fuzzy. We then lookup the Neural memory from that layer onwards to determine if the text is spam. In this setup, we assume documents consisted upto 10 images and represented as bag of images. In our experiments, we utilized pre-trained CNN feature extractors and could handle perfect classification up to vocabulary size of $10,000$ with $5,000$ to be spam word-image.

Long Tail Image Classification In addition to enlarging the model capacity, external memory helps networks memorize few-shot examples in training data. Sketch based memory augmented ResNet gives a significant boost to the ImageNet-LT classification task (Liu et al., 2019). ImageNet-LT dataset (Liu et al., 2019) is a long-tailed version of the ImageNet dataset (Deng et al., 2009) that consists of training image classes with many (> 100), medium ($\leq 100 \,\&\, > 20$) and few (< 20) number of images, and open set image classes (i.e. images that should be classified as "unseen"). Our memory augmented model consists of the ResNet-10 network (He et al., 2016a) as backbone feature extractor and a memory-augmented dot product classification layer. Despite simplicity, our model achieves better or similar performance compared to both memoryless models (Plain model (He et al., 2016a), Lifted Loss (Oh Song et al., 2016), etc) and memory-based models (OLTR (Liu et al., 2019)), see Tab. 1.

Patent Assignee Resolution In patent assignee resolution, the task is to map a large number of mentions (aliases of assignee names) to the correct assignee names. Each mention is a noised version of the true assignee. For example, assignee "ABC co. LLC" can be mentioned as "ABC co., LLC" or "abc llc". The problem is thus similar to the fuzzy-key value lookup problem. We take the patent assignee mention data (of Economic Research, 2010 (accessed February 3, 2020), and keep assignees with no less than 5 mentions in our train and test data. Train dataset contains 9696 unique assignees and 57793 mentions, and test dataset contains the same assignees and 16842 mentions. Baseline model is a character n-gram bidirectional LSTM (Hochreiter and Schmidhuber, 1997) model (with a small version with 160 dimensional output in LSTM and a large version with 320 dimensional output in LSTM). SketchMem model shares the same network structure as the small baseline, with SketchMem augmented only at the input embedding layer. We observe SketchMem boosts accuracy, especially when baseline is constrained to have a small embedding vocabulary size, see Table-2. We also observed that a k Nearest Neighbor (kNN) model achieves a 92.5% with TF-IDF features as input (not in the table). The gap between

Backbone Net	**closed-set setting**				**open-set setting**			
ResNet-10	> 100	$\leqslant 100$ & > 20	< 20		> 100	$\leqslant 100$ & > 20	< 20	
Methods	**Many**	**Medium**	**Few**	**Overall**	**Many**	**Medium**	**Few**	**F-meas**
Plain Model [1]	40.9	10.7	0.4	20.9	40.1	10.4	0.4	0.295
Lifted Loss [2]	35.8	30.4	17.9	30.8	34.8	29.3	17.4	0.374
Focal Loss [3]	36.4	29.9	16.0	30.5	35.7	29.3	15.6	0.371
Range Loss [4]	35.8	30.3	17.6	30.7	34.7	29.4	17.2	0.373
+ OpenMax [5]	-	-	-	-	35.8	30.3	17.6	0.368
FSLwF [6]	40.9	22.1	15	28.4	40.8	21.7	14.5	0.347
OLTR [7]	43.2	35.1	18.5	35.6	41.9	33.9	17.4	**0.474**
SketchMem (Ours)	**44.5**	**36.9**	**18.7**	**37.2**	**43.1**	**35.7**	**18.1**	0.440

Table 1: Top-1 classification accuracy on ImageNet-LT compared to methods presented in [1]: He et al. (2016a) , [2]: Oh Song et al. (2016), [3]: Lin et al. (2017), [4]: Zhang et al. (2017), [5]: Bendale and Boult (2016), [6]: Gidaris and Komodakis (2018), [7]: Liu et al. (2019).

$\lvert\mathcal{V}\rvert$	Small baseline	Large baseline	SketchMem
2048	39.48%	41.23%	74.71%
8192	65.09%	66.78%	74.87%

Table 2: Accuracy for the patent assignee resolution, for different embedding vocabulary size $|\mathcal{V}|$. Small baseline has about 82 millon FLOPs/sample, large baseline has about 246 million FLOPs/sample, and SketchMem model has about 90 million FLOPs /sample.

kNN and SketchMem could be due to the *approximate* nearest neighbor that SketchMem performs, and will be studied in future work.

Compact BERT models Pretrained masked language models such as BERT has shown impressive performance for a range of NLP tasks (Devlin et al., 2018). Augmenting BERT models with SketchMem provides an computationally efficient way of increasing the model capacity. We examined the effect of memory augmentation for compact BERT models with 2, 4, 8 and 12 transformer layers. In this experiment, SketchMem is augmented to the attention layer in every transformer layer. For an input token embedding sequence $\{x_1, x_2, ..., x_n\}$, the i-th output of the SketchMem-augmented attention layer is $\text{atten}(x_1, ..., x_i, ..., x_n) + \text{sketchmem}(x_i)$, where $\text{atten}(x_1, ..., x_i, ..., x_n)$ is the original attention layer output and $\text{sketchmem}(x_i)$ is the SketchMem output. Table 3 compares 5 compact BERT models with their SketchMem augmented counterparts. The models are pretrained on the wiki and books dataset (Devlin et al., 2018). BERT-tiny, BERT-mini, BERT-small and BERT-medium are pretrained for 5 epochs, and BERT-base is pretrained for 15 epochs. All the models are then finetuned for 3 epochs for downstream tasks. For all 5 models, the SketchMem uses 3 hash functions and 1024 (2^{10}) memory buckets. We observed that SketchMem augmented models boost performance for both masked language modeling and downstream tasks, with a tiny additional computational cost.

5 Discussion and Related Works

We proposed a uniform framework for augmenting a neural network with external memory to improve the capacity of the network independent of the problem. The neural memory uses LSH and sketching to access information in the memory during inference – the memory contents are trained using a simple problem independent mechanism during training. We demonstrated theoretically and empirically that adding neural memory enables bounded sized neural networks to learn a wide variety of problems that may need to store large amounts of data and thus hard to solve by bounded sized neural networks.

The idea of augmenting a neural network with memory has been suggested in several papers like LSTM (Hochreiter and Schmidhuber, 1997), NeuralRAM (Kurach et al., 2016), NeuralStack (Grefenstette et al., 2015), NeuralGPU (Kaiser and Sutskever, 2015), NTM (Graves et al., 2014), DNC (Graves et al., 2016). Their purpose has, however, been different – most of these prior work have been focused around augmenting the neural network with a working memory for solving one instance of the problem. On the other hand, we propose to use SketchMem in a persistent fashion as an episodic memory, i.e. use content of memory across different problem instances.

Several works store certain statistics of the training data to improve performance – however they lack the kind of theoretical guarantees we provide. In particular, Khandelwal et al. (2019) explicitly stores the hidden representation of the entire training dataset to improve language model predictions and cloze test. Similarly, REALM (Guu et al., 2020) and RAG (Lewis et al., 2020) retrieve from an external datastore for each prediction, however, the designs are very task specific, unlike SketchMem which is a generic framework. Another use case of retaining parts of training data has been to prevent catastrophic forgetting McCloskey and Cohen (1989); Ratcliff (1990). Techniques have been developed to ensure the distribution of stored examples is representative of the distribution of true exam-

Model	BERT-tiny	BERT-mini	BERT-small	BERT-medium	BERT-base
FLOPS/token (rel.)	1.000	8.000	32.00	64.00	216
+ SketchMem	1.029	8.117	32.23	64.47	217.05
MLM (Acc)	31.55	47.50	55.39	58.56	65.82
+ SketchMem	$32.46^{(+2.88\%)}$	$49.67^{(+4.57\%)}$	$56.96^{(+2.83\%)}$	$60.27^{(+2.92\%)}$	$67.02^{(+1.82\%)}$
MNLI (Acc)	63.40	72.63	76.19	78.63	82.19
+ SketchMem	$65.63^{(+3.51\%)}$	$73.72^{(+1.50\%)}$	$79.27^{(+4.04\%)}$	$81.79^{(+4.02\%)}$	$86.10^{(+4.76\%)}$
SQuAD 1.1 (f1)	12.99	65.88	77.57	82.43	88.32
+ SketchMem	$42.31^{(+225\%)}$	$69.77^{(+5.90\%)}$	$80.41^{(+3.66\%)}$	$84.89^{(+2.98\%)}$	$89.31^{(+1.12\%)}$
SQuAD 2.0 (f1)	50.09	59.17	65.09	68.90	76.56
+ SketchMem	$51.70^{(+3.21\%)}$	$60.20^{(+1.74\%)}$	$65.97^{(+1.35\%)}$	$70.73^{(+2.65\%)}$	$78.31^{(+2.29\%)}$

Table 3: Compact BERT models. Denote the number of transformer layers as L, the hidden embedding size as H, and fix the number of self-attention heads to be $H/64$ and feedforward size to be $4H$. The 5 models have the following parameters: BERT-tiny (L=2, H=128), BERT-mini (L=4, H =256), BERT-small (L=4, H=512), BERT-medium (L=8, H=512) and BERT-base (L=12, H=768). SketchMem is fixed to have 3 hash functions and 1024 memory buckets. The FLOPS row counts flops per token for the feedforward network and the feedforward network + SketchMem. Note in the MNLI task, BERT-small + SketchMem outperforms BERT-medium, while it uses much less flops than BERT-medium.

ples; (Isele and Cosgun, 2018; de Masson d'Autume et al., 2019) and the episodic memory is used to either constraint the gradient updates (Lopez-Paz and Ranzato, 2017; Chaudhry et al., 2019), locally adapt the base model to a new test example (Vinyals et al., 2016; Sprechmann et al., 2018), or for experience replay (Wang et al., 2019; de Masson d'Autume et al., 2019). Another work (Sukhbaatar et al., 2019) augments the transformer with a persistent embedding table that may be attended to in addition to the input tokens. Product key memory (Lample et al., 2019) augments the dense layers in transformer with a large matrix-memory and uses product quantization to enable fast look-up. However, we would like to point out all of these methods require custom design for selection and use of past training data as opposed to SketchMem.

As the name suggests, SketchMem tries to store sketches of useful training examples. The idea of sketching to represent and store the neural processing of a complex input has been studied theoretically in Ghazi et al. (2019); Panigrahy (2019) previously. They propose the idea of using a recursive sketch to store essential properties of a complex input so that underlying objects can be compared just by sketch similarity. Our sketching method can be viewed as a simplification of those methods. Sketching has been used to get a condensed representation of several types of objects including documents (Broder, 1997), large graphs(Das Sarma et al., 2010), sets and vectors (Clarkson and Woodruff, 2009).

Finally, the idea of combining LSH with neural networks habe been explored before, but most works have suggested it as a method to speed up the training instead of increasing its memory capacity. For example Spring and Shrivastava (2017) and Kitaev et al. (2020) use LSH to find active neurons in a layer quickly by identifying those weight vectors in a layer that have a high inner product with the input and dropping the rest during the forward=backward pass (similar to random dropout but chosen carefully). This allows training of transformers with self-attention for very long sequences by restricting attention to only important candidates.

References

Alexandr Andoni and Ilya Razenshteyn. Optimal data-dependent hashing for approximate near neighbors. In *Proceedings of the forty-seventh annual ACM symposium on Theory of computing*, pages 793–801, 2015.

Alexandr Andoni, Piotr Indyk, Huy L Nguyen, and Ilya Razenshteyn. Beyond locality-sensitive hashing. In *Proceedings of the twenty-fifth annual ACM-SIAM symposium on Discrete algorithms*, pages 1018–1028. SIAM, 2014.

Alexandr Andoni, Piotr Indyk, Thijs Laarhoven, Ilya Razenshteyn, and Ludwig Schmidt. Practical and optimal lsh for angular distance. In *Advances in neural information processing systems*, pages 1225–1233, 2015.

Sanjeev Arora, Simon Du, Wei Hu, Zhiyuan Li, and Ruosong Wang. Fine-Grained Analysis of Optimization and Generalization for Overparameterized Two-Layer Neural Networks. In *International Conference on Machine Learning*, pages 322–332, May 2019.

Abhijit Bendale and Terrance E Boult. Towards open set deep networks. In *Proceedings of the IEEE con-*

ference on computer vision and pattern recognition, pages 1563–1572, 2016.

Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

Andrei Z Broder. On the resemblance and containment of documents. In *Proceedings. Compression and Complexity of SEQUENCES 1997 (Cat. No. 97TB100171)*, pages 21–29. IEEE, 1997.

Moses S Charikar. Similarity estimation techniques from rounding algorithms. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, pages 380–388, 2002.

Arslan Chaudhry, Marcus Rohrbach, Mohamed Elhoseiny, Thalaiyasingam Ajanthan, Puneet K Dokania, Philip HS Torr, and Marc'Aurelio Ranzato. Continual learning with tiny episodic memories. *arXiv preprint arXiv:1902.10486*, 2019.

Kenneth L Clarkson and David P Woodruff. Numerical linear algebra in the streaming model. In *Proceedings of the forty-first annual ACM symposium on Theory of computing*, pages 205–214, 2009.

Atish Das Sarma, Sreenivas Gollapudi, Marc Najork, and Rina Panigrahy. A sketch-based distance oracle for web-scale graphs. In *Proceedings of the third ACM international conference on Web search and data mining*, pages 401–410, 2010.

Cyprien de Masson d'Autume, Sebastian Ruder, Lingpeng Kong, and Dani Yogatama. Episodic memory in lifelong language learning. In *Advances in Neural Information Processing Systems*, pages 13122–13131, 2019.

J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

Simon S. Du, Xiyu Zhai, Barnabás Póczos, and Aarti Singh. Gradient descent provably optimizes over-parameterized neural networks. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. URL `https://openreview.net/forum?id=S1eK3i09YQ`.

Joe Ellis, Jeremy Getman, Dana Fore, Neil Kuster, Zhiyi Song, Ann Bies, and Stephanie M Strassel. Overview of linguistic resources for the tac kbp 2015 evaluations: Methodologies and results.

Michael Eysenck. *Attention and arousal: Cognition and performance*. Springer Science & Business Media, 2012.

Vitaly Feldman. Does learning require memorization? a short tale about a long tail, 2019.

Badih Ghazi, Rina Panigrahy, and Joshua Wang. Recursive sketches for modular deep learning. In *International Conference on Machine Learning*, pages 2211–2220, 2019.

Spyros Gidaris and Nikos Komodakis. Dynamic few-shot visual learning without forgetting. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4367–4375, 2018.

Alex Graves, Greg Wayne, and Ivo Danihelka. Neural turing machines. *arXiv preprint arXiv:1410.5401*, 2014.

Alex Graves, Greg Wayne, Malcolm Reynolds, Tim Harley, Ivo Danihelka, Agnieszka Grabska-Barwińska, Sergio Gómez Colmenarejo, Edward Grefenstette, Tiago Ramalho, John Agapiou, et al. Hybrid computing using a neural network with dynamic external memory. *Nature*, 538(7626):471–476, 2016.

Edward Grefenstette, Karl Moritz Hermann, Mustafa Suleyman, and Phil Blunsom. Learning to transduce with unbounded memory. In *Advances in neural information processing systems*, pages 1828–1836, 2015.

Kelvin Guu, Kenton Lee, Zora Tung, Panupong Pasupat, and Ming-Wei Chang. Realm: Retrieval-augmented language model pre-training, 2020.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016a.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016b.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

David Isele and Akansel Cosgun. Selective experience replay for lifelong learning. In *Thirty-second AAAI conference on artificial intelligence*, 2018.

Herve Jegou, Matthijs Douze, and Cordelia Schmid. Product quantization for nearest neighbor search. *IEEE transactions on pattern analysis and machine intelligence*, 33(1):117–128, 2010.

Łukasz Kaiser and Ilya Sutskever. Neural gpus learn algorithms. *arXiv preprint arXiv:1511.08228*, 2015.

Urvashi Khandelwal, Omer Levy, Dan Jurafsky, Luke Zettlemoyer, and Mike Lewis. Generalization through memorization: Nearest neighbor language models. *arXiv preprint arXiv:1911.00172*, 2019.

Nikita Kitaev, Łukasz Kaiser, and Anselm Levskaya. Reformer: The efficient transformer. *arXiv preprint arXiv:2001.04451*, 2020.

Michael Krivelevich and Benny Sudakov. The largest eigenvalue of sparse random graphs. *Combinatorics, Probability and Computing*, 12(1):61–72, 2003.

Karol Kurach, Marcin Andrychowicz, and Ilya Sutskever. Neural random-access machines. In *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016.

Guillaume Lample, Alexandre Sablayrolles, Marc'Aurelio Ranzato, Ludovic Denoyer, and Hervé Jégou. Large memory layers with product keys. In *Advances in Neural Information Processing Systems*, pages 8548–8559, 2019.

Patrick Lewis, Ethan Perez, Aleksandara Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. *arXiv preprint arXiv:2005.11401*, 2020.

Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *Proceedings of the IEEE international conference on computer vision*, pages 2980–2988, 2017.

Ziwei Liu, Zhongqi Miao, Xiaohang Zhan, Jiayun Wang, Boqing Gong, and Stella X Yu. Large-scale long-tailed recognition in an open world. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2537–2546, 2019.

David Lopez-Paz and Marc'Aurelio Ranzato. Gradient episodic memory for continual learning. In *Advances in Neural Information Processing Systems*, pages 6467–6476, 2017.

Michael McCloskey and Neal J Cohen. Catastrophic interference in connectionist networks: The sequential learning problem. In *Psychology of learning and motivation*, volume 24, pages 109–165. Elsevier, 1989.

Tomáš Mikolov, Martin Karafiát, Lukáš Burget, Jan Černockỳ, and Sanjeev Khudanpur. Recurrent neural network based language model. In *Eleventh annual conference of the international speech communication association*, 2010.

National Bureau of Economic Research. *Patent Data Project*, 2010 (accessed February 3, 2020). `https://sites.google.com/site/patentdataproject/Home/downloads`.

Hyun Oh Song, Yu Xiang, Stefanie Jegelka, and Silvio Savarese. Deep metric learning via lifted structured feature embedding. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4004–4012, 2016.

Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*, 2016.

Rina Panigrahy. Entropy based nearest neighbor search in high dimensions. In *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithm*, SODA '06, page 1186–1195, USA, 2006. Society for Industrial and Applied Mathematics. ISBN 0898716055.

Rina Panigrahy. How does the mind store information?, 2019.

Martin Raab and Angelika Steger. "balls into bins"—a simple and tight analysis. In *International Workshop on Randomization and Approximation Techniques in Computer Science*, pages 159–170. Springer, 1998.

Roger Ratcliff. Connectionist models of recognition memory: constraints imposed by learning and forgetting functions. *Psychological review*, 97(2):285, 1990.

Mark Rudelson and Roman Vershynin. Smallest singular value of a random rectangular matrix. *Communications on Pure and Applied Mathematics: A Journal Issued by the Courant Institute of Mathematical Sciences*, 62(12):1707–1739, 2009.

Pablo Sprechmann, Siddhant M Jayakumar, Jack W Rae, Alexander Pritzel, Adria Puigdomenech Badia, Benigno Uria, Oriol Vinyals, Demis Hassabis, Razvan Pascanu, and Charles Blundell. Memory-based parameter adaptation. *arXiv preprint arXiv:1802.10542*, 2018.

Ryan Spring and Anshumali Shrivastava. Scalable and sustainable deep learning via randomized hashing. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 445–454, 2017.

Sainbayar Sukhbaatar, Edouard Grave, Guillaume Lample, Hervé Jégou, and Armand Joulin. Augmenting self-attention with persistent memory. *CoRR*, abs/1907.01470, 2019. URL `http://arxiv.org/abs/1907.01470`.

Linh V Tran, Van H Vu, and Ke Wang. Sparse random graphs: Eigenvalues and eigenvectors. *Random Structures & Algorithms*, 42(1):110–134, 2013.

Oriol Vinyals, Charles Blundell, Timothy Lillicrap, Daan Wierstra, et al. Matching networks for one shot learning. In *Advances in neural information processing systems*, pages 3630–3638, 2016.

Hong Wang, Wenhan Xiong, Mo Yu, Xiaoxiao Guo, Shiyu Chang, and William Yang Wang. Sentence embedding alignment for lifelong relation extraction. *arXiv preprint arXiv:1903.02588*, 2019.

Manzil Zaheer, Sashank Reddi, Devendra Sachan, Satyen Kale, and Sanjiv Kumar. Adaptive methods for nonconvex optimization. In *Advances in Neural Information Processing Systems 31*, pages 9793–9803, 2018.

Xiao Zhang, Zhiyuan Fang, Yandong Wen, Zhifeng Li, and Yu Qiao. Range loss for deep face recognition with long-tailed training data. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 5409–5418, 2017.