

# METEOR: Learning Memory and Time Efficient Representations from Multi-modal Data Streams\*

Amila Silva, Shanika Karunasekera, Christopher Leckie and Ling Luo

School of Computing and Information Systems

The University of Melbourne

Parkville, Victoria, Australia

{amil.silva@student., karus@, caleckie@, ling.luo@}unimelb.edu.au

## ABSTRACT

Many learning tasks involve multi-modal data streams, where continuous data from different modes convey a comprehensive description about objects. A major challenge in this context is how to efficiently interpret multi-modal information in complex environments. This has motivated numerous studies on learning unsupervised representations from multi-modal data streams. These studies aim to understand higher-level contextual information (e.g., a Twitter message) by jointly learning embeddings for the lower-level semantic units in different modalities (e.g., text, user, and location of a Twitter message). However, these methods directly associate each low-level semantic unit with a continuous embedding vector, which results in high memory requirements. Hence, deploying and continuously learning such models in low-memory devices (e.g., mobile devices) becomes a problem. To address this problem, we present METEOR, a novel **ME**emory and **Time** Efficient **Online Representation** learning technique, which: (1) learns compact representations for multi-modal data by sharing parameters within semantically meaningful groups and preserves the domain-agnostic semantics; (2) can be accelerated using parallel processes to accommodate different stream rates while capturing the temporal changes of the units; and (3) can be easily extended to capture implicit/explicit external knowledge related to multi-modal data streams. We evaluate METEOR using two types of multi-modal data streams (i.e., social media streams and shopping transaction streams) to demonstrate its ability to adapt to different domains. Our results show that METEOR preserves the quality of the representations while reducing memory usage by around 80% compared to the conventional memory-intensive embeddings.

## ACM Reference Format:

Amila Silva, Shanika Karunasekera, Christopher Leckie and Ling Luo. 2020. METEOR: Learning Memory and Time Efficient Representations from Multi-modal Data Streams. In *CIKM '20: ACM Conference on Information and Knowledge Management, October 19–23, 2020, Galway, Ireland*. ACM, New York, NY, USA, 11 pages. <https://doi.org/10.1145/nnnnnnn.nnnnnnn>

\* Accepted as a conference paper at CIKM 2020

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from [permissions@acm.org](https://permissions.acm.org).

CIKM '20, October 19–23, 2020, Galway, Ireland

© 2020 Association for Computing Machinery.

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM...\$15.00

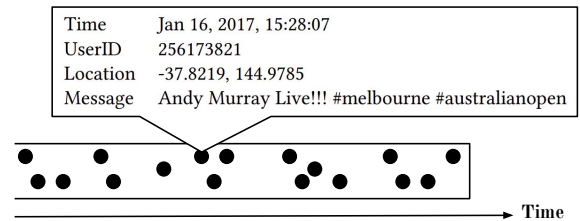
<https://doi.org/10.1145/nnnnnnn.nnnnnnn>

## 1 INTRODUCTION

Unsupervised representation learning [2, 28] has recently become a rapidly growing direction in machine learning due to its ability to: (1) exploit the availability of unlabeled data; and (2) associate the underlying factors behind data effortlessly. For a given domain, conventional representational learning techniques learn low-dimensional vectors (i.e., embeddings) for low-level data units (e.g., words in a language) such that these representations capture the underlying semantics of the particular domain in a task-independent manner. For example, the language modelling techniques in [10, 13] learn embeddings for words or characters by considering them as the low-level units<sup>1</sup> of a language, from which higher-level structures (e.g., phrases and sentences) can be constructed and understood. Subsequently, these representations serve as features to solve different application-level problems. Lately, it has been empirically proven that these representations yield a significant performance boost for many downstream tasks in domains such as Natural Language Processing [3, 13] and Computer Vision [8, 21].

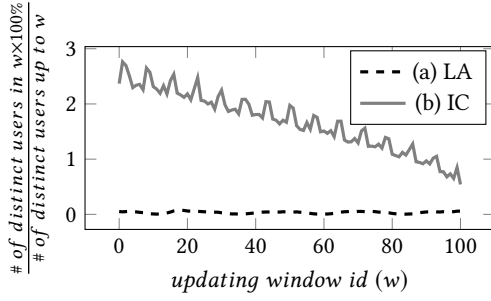
This paper focuses on representation learning techniques for multi-modal data streams: techniques to learn representations from records with different types of low-level units (i.e., attributes) in an online fashion to capture temporal changes of the units while preserving the relationships between different modalities. For example, a geo-tagged Twitter stream (see Figure 1) is such a multi-modal data stream, in which each record has multiple types of attributes, such as its location, user, timestamp, and text content. Although there has been previous work [19, 27] on this problem, our work aims to address the following research gaps in the literature.

**Research Gaps.** First, conventional embedding learning techniques are memory-intensive as they assign an independent embedding vector to each low-level unit. If such techniques are extended to embed attributes with a growing number of distinct low-level units, such as users and words, the amount of memory required to store all the embedding vectors becomes a major overhead. For



**Figure 1: Geo-tagged Twitter Stream as an example of a multi-modal data stream**

<sup>1</sup>Such low-level units in different domains are referred to as “units” in this paper.



**Figure 2: The fraction of different users that appear within an updating window of length  $\Delta W$  compared to the total users seen up to the particular window: (a) LA, a Geo-tagged Twitter Stream ( $\Delta W = 1$  hour); and (b) IC, a Shopping Transaction Stream ( $\Delta W = 1$  day). More details about LA and IC are provided in Section 5**

example, consider a system that learns embeddings using a shopping transaction record stream, which consists of two types of low-level units: (1) 1 million distinct users; and (2) 1 million different products. If a 300-dimensional vector is assigned to each unit, the total memory requirement for the embeddings becomes<sup>2</sup>  $(1,000,000 + 1,000,000)(units) \times 300(dimensions) \times 4(bytes) \approx 2.4GB$ . Moreover, some multi-modal streams can have more than two modalities. If we consider a geo-tagged Twitter stream as an example, there are four modalities, namely: location, text, user, and timestamp. Hence, the total memory required for such a stream can be substantially higher than the example above. Thus, the application of conventional embedding learning approaches for multi-modal streams becomes problematic, particularly on limited memory platforms. There are several previous works (compared in Section 2) on learning compact memory-efficient representations. However, almost all these methods are not well suited to data streams [4, 18], or they are specific to a particular domain (e.g., the technique proposed in [16] is specific to natural language modelling). Thus, this paper proposes **a domain-agnostic and memory-efficient representation learning technique to work with data streams**.

Second, the processing time per record of online learning techniques should meet at least the rate of the stream (i.e., 1/average records per unit time) to update models in a timely manner. Parallel-processing architectures can be used to meet this requirement when working with data streams. However, this problem is not well studied in the context of online representation learning, possibly due to the memory-intensive nature of conventional embeddings. With memory-efficient representations, it is feasible to adopt parallel-processing to reduce the time complexity of online representation learning techniques. We propose **a decomposable objective function to learn memory-efficient representations from streams, which allows the flexibility to assign more parallel processes (with different memory capacities) for computationally expensive steps in METEOR**.

<sup>2</sup>Here we assume that the values in the embeddings are represented as single-precision float numbers (4 bytes per value).

Third, some of the attributes in multi-modal data streams show specific relationships or behaviours (either explicit or implicit). For example, the products that appear in a shopping transaction stream can be grouped using higher-level product categories. Although such explicit relationships have been exploited in previous works [26] to improve the quality of the representations, they are not well-studied for the task of making representations compact. Also, for a given attribute of a multi-modal stream, we observe that a small fraction of units of the attribute appear during a short period of the stream compared to the total number of distinct units of the particular attribute. For example, the fraction of users appear in an updating window is less than 3% of the total users for two multi-modal streams as shown in Figure 2. Likewise, the attributes show specific relationships and behaviours in multi-modal data streams. Our model is designed in a manner to **exploit such explicit/implicit relationships in multi-modal data streams**.

**Contributions.** In this paper, we propose METEOR, a novel compact representation learning technique using multi-modal data streams, which:

- learns compact online representations for multi-modal data units by sharing the parameters (i.e., basis vectors) inside the semantically meaningful groupings of the units. Also, METEOR is domain-agnostic, thus yielding consistent results with multi-modal streams from different domains;
- proposes an architecture (METEOR-PARALLEL) to accelerate the learning of METEOR, which can learn embeddings of the same quality at twice the speed of METEOR; and
- can be easily extended to exploit explicit knowledge sources related to multi-modal data units by defining parameter sharing groups (implicitly defined if there is no such explicit knowledge source) based on the particular knowledge sources. Our results show that METEOR can further improve the quality of the compact embeddings by including explicit knowledge sources.

## 2 RELATED WORK

Although work on learning compressed coding systems began in the 1950's, such as error correction codes [7], and Hoffman codes [9], there has been recent progress in representation learning techniques [13, 15] that encode each low-level unit with a continuous embedding vector. In such approaches, the number of parameters in the embeddings grows linearly with the number of low-level units. As a result, learning compact representations has become a popular research problem and recently been addressed by many previous works. Almost all the previous works on learning memory-efficient representations can be divided into two categories: (1) compositional embedding learning techniques; and (2) data compression techniques.

**(1) Compositional embedding learning techniques.** In these techniques, a set of basis vectors are learned, which are shared between all low-level units. Then the final embedding for a given unit is taken as a composition (e.g., linear combination) of the basis vectors. These techniques mainly differ from each other in the way the basis vectors are composed to generate final representations. METEOR also belongs to this category.

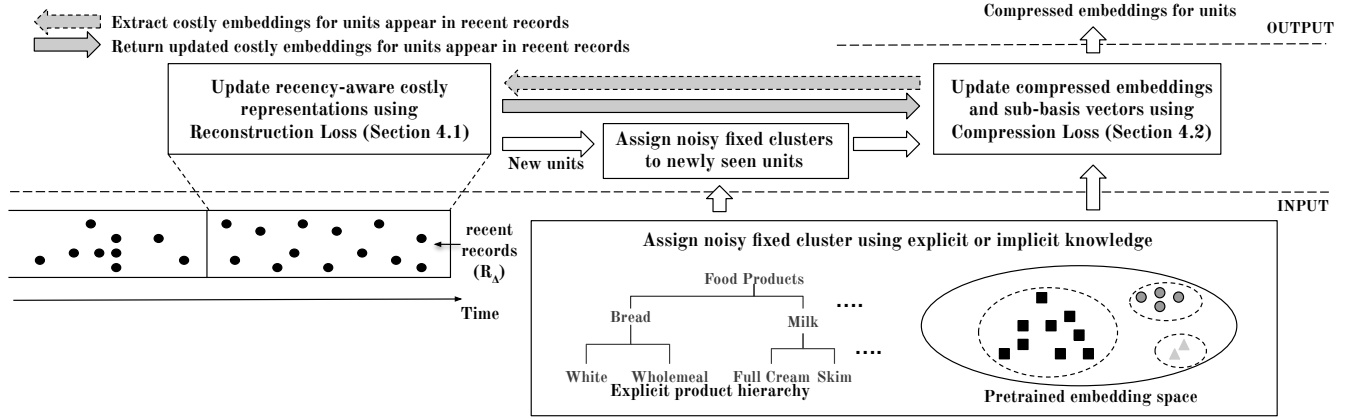


Figure 3: Overview of METEOR

In some works [17, 24], predefined hashing functions (e.g., division hashing and tree based hashing) are used to assign each low-level unit to one of the basis vectors. *However, these approaches do not consider the semantics of the units when mapping to the basis vectors, thus they may blindly map vastly different units to the same basis vector.* This can lead to substantial loss of information and deterioration of the embeddings. To mitigate that, other recent works explore domain-specific sub-units with smaller vocabularies to define basis vectors. For example, in [10, 16, 23], the basis vectors are defined using characters and sub-words in a language. Then hash functions are defined to automatically map words (or texts) to pre-defined bases, according to which the vectors are composed. However, such approaches may not be scalable for other domains as they use language-specific semantics when defining their compositions. In contrast, METEOR exploits the semantics of the low-level units in a domain-agnostic manner.

To learn task-independent compact embeddings, a few previous works [4, 18] attempt to jointly learn basis vectors and discrete codes for each unit, which defines the composition of the basis vectors. However, the main challenge of learning discrete codes to define composition is that they cannot be directly optimized via SGD like other parameters. In [4, 18], the discrete encodings are relaxed using continuous relaxation techniques such as the Tempering-Softmax trick and Gumbel-Softmax trick. In [22], a similar technique has been proposed, which divides the aforementioned non-differentiable objective function into several solvable sub-problems, and sequentially solves each sub-problem. All these techniques require continuous costly embeddings (e.g., pretrained word embeddings) to be stored to guide their learning process. Also, we empirically observe that the latter is computationally expensive to learn in an online fashion. Thus, these techniques are not suitable for data streams, where the learning happens incrementally. In contrast, METEOR learns sparse continuous code vectors along with basis vectors, which could be trained in an online fashion. Also, to the best of our knowledge, METEOR is the first work on learning compact online embeddings using multi-modal data streams.

**(2) Data compression techniques.** In addition, some previous works [1, 12, 14] adopt data compression techniques (e.g., quantization, dimension reduction, sparse coding, quantum entanglement)

to reduce the memory requirement to store embeddings. However, all these approaches linearly increase the size of the embedding table with the number of distinct units in the system, whereas METEOR introduces a small overhead with respect to the total number of distinct units.

### 3 PROBLEM STATEMENT

Let  $R = \{r_1, r_2, r_3, \dots, r_n, \dots\}$  be a continuous stream of records that arrive in chronological order. Each record  $r \in R$  is a tuple  $\langle a_0^r, a_1^r, \dots, a_N^r \rangle$ , where  $a_i^r$  is the  $i^{th}$  attribute of  $r$  and  $N$  denotes the number of attributes of  $r$ .

Our problem is to learn embeddings for all possible units in each attribute, denoted as  $A_i (= \bigcup_{r \in R} \{a_i^r\})$  for the  $i^{th}$  attribute, such that the embedding  $v_x$  of a unit  $x \in A_i$ :

- (1) is a  $d$ -dimensional vector ( $d \ll \sum_{i=0}^N |A_i|$ ), where  $|A_i|$  is the number of different units in  $A_i$ ;
- (2) preserves the co-occurrences of the attributes;
- (3) is continuously updated as new records ( $R_\Delta$ ) arrive to incorporate the latest information;
- (4) is memory efficient with a memory complexity  $\ll O(d)$ .

Consider a shopping transaction stream as an example for  $R$ , in which each record (i.e., transaction)  $r$  can be characterized using a tuple consisting of three attributes  $\langle t^r, u^r, p^r \rangle$ , where: (1)  $t^r$  is the transaction time of  $r$ ; (2)  $u^r$  is the user id of  $r$ ; and (3)  $p^r$  is the set of products in the shopping basket of  $r$ . This work aims to jointly learn recency-aware vector representations for all possible units in each attribute (e.g., discretized<sup>3</sup> timestamps, products, and users in  $R$ ) such that the co-occurring units have similar embeddings.

### 4 METEOR

**Overview.** For a given attribute (e.g., products) in a multi-modal data stream (e.g., shopping transaction stream), METEOR groups all the possible low-level units (e.g., white bread and skim milk) of the particular attribute into a set of semantically meaningful clusters (i.e., *noisy fixed clusters*). Then for each cluster, a set of basis vectors are trained to learn the *costly embeddings* (i.e., conventional

<sup>3</sup>Continuous attributes such as timestamps should be discretized to make them feasible for embedding

memory intensive embeddings) for the units in the particular cluster as a composition of the corresponding basis vectors. For each unit, the composition of the basis vectors (denoted as the *compressed embedding*) is defined as a sparse continuous vector.

Despite learning the compressed embeddings of the units directly, METEOR follows a computationally efficient sequential approach as shown in Figure 3. For a given set of recent records  $R_\Delta$ , METEOR initially extracts the costly embeddings for the units that appeared in  $R_\Delta$ , then updates the extracted costly embeddings using  $R_\Delta$  to incorporate the recent information. Subsequently, the recency-aware costly embeddings are used to update the corresponding compressed embeddings and basis vectors. This section discusses in detail how these steps are performed incrementally.

#### 4.1 Learning Costly Embeddings

The desired embedding space of METEOR should be able to predict any attribute of a record given the other attributes (i.e., to preserve the first-order co-occurrences in records). Thus, the costly embeddings in METEOR are learned to recover the attributes of records as much as possible, which is formally elaborated as follows.

For a given record  $r$ , the embeddings are learned such that the units (e.g., product or user in a shopping transaction) of  $r$  can be recovered by looking at  $r$ 's other units. Formally, we model the likelihood for the task of recovering unit  $x \in r$  given the other units  $r_{-x}$  of  $r$  as:

$$p(x|r_{-x}) = \exp(s(x, r_{-x})) / \sum_{y \in X} \exp(s(y, r_{-x})) \quad (1)$$

where  $X$  is the type (e.g., product or user in a shopping transaction) of  $x$ , and  $s(x, r_{-x})$  is the similarity score between  $x$  and  $r_{-x}$ . We define the  $s(x, r_{-x})$  as  $s(x, r_{-x}) = v_x^T h_x$  where  $h_x$  is mean embedding of  $r$ 's units except  $x$ .

Then, the final loss function for the attribute recovery task is the negative log likelihood of recovering all the attributes of the records in the current buffer  $B$ :

$$O_{R_\Delta} = - \sum_{r \in R_\Delta} \sum_{x \in r} p(x|r_{-x}) \quad (2)$$

The objective function above is approximated using negative sampling (proposed in [13]) for efficient optimization using stochastic gradient descent (SGD). Then for a selected record  $r$  and unit  $x \in r$ , the loss function is:

$$L_{recon} = -\log(\sigma(s(x, r_{-x}))) - \sum_{n \in N_x} \log(\sigma(-s(n, r_{-x}))) \quad (3)$$

where  $\sigma(z) = \frac{1}{1+\exp(-z)}$  and  $N_x$  is the set of randomly selected negative units that have the type of  $x$ .

**Adaptive Optimization.** Since the loss function is incrementally optimized using a stream, only the recent records in the stream are used to update the embeddings. Hence, we adopt a novel adaptive strategy to optimize the loss function in Equation 3 while alleviating overfitting to the recent records as follows.

For each record  $r$ , we compute the intra-agreement  $\Psi_r$  of  $r$ 's attributes as:

$$\Psi_r = \frac{\sum_{x, y \in r, x \neq y} \sigma(v_x^T v_y)}{\sum_{x, y \in r, x \neq y} 1} \quad (4)$$

Then the adaptive learning rate of  $r$  is calculated as,

$$lr_r = \exp(-\tau \Psi_r) * \eta \quad (5)$$

where  $\eta$  denotes the standard learning rate and  $\tau$  controls the importance given to  $\Psi_r$ . If the representations have already overfitted to  $r$ , then  $\Psi_r$  takes a higher value. Consequently, a low learning rate is assigned to  $r$  to avoid overfitting. In addition, the learning rate for each unit  $x$  in  $r$  is further weighted using the approach proposed in AdaGrad [5] to alleviate overfitting to frequent items. Then, the update for the  $v_x$  at the  $t^{th}$  timestep is:

$$v_x^{t+1} = v_x^t - \frac{lr_r}{\sqrt{\sum_{j=0}^{t-1} (\frac{\partial L}{\partial v_x})_j^2} + \epsilon} \left( \frac{\partial L}{\partial v_x} \right)_t \quad (6)$$

Our experimental results verify that the proposed optimization technique to accommodate online learning yields comparable (sometimes even superior) results compared to state-of-the-art sampling-based approaches<sup>4</sup> without storing any historical records.

#### 4.2 Learning Compressed Embeddings

The aforementioned framework learns an independent embedding vector for each unit, and is thus memory inefficient. To alleviate this problem, METEOR takes the following steps:

- Step 1: Define a set of clusters (i.e., noisy fixed clusters) for a given attribute such that each unit of the particular attribute belongs to a single cluster.
- Step 2: Assign a set of basis vectors for each cluster.
- Step 3: Impose an additional constraint on costly embeddings of the units such that they are linear combinations of the basis vectors of the corresponding noisy fixed clusters.

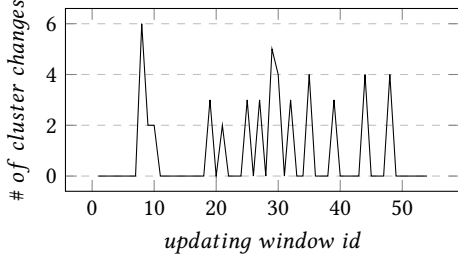
Then, the compressed embedding  $\hat{v}_x$  of a given unit  $x$  is the set of weights related to the corresponding basis vectors (i.e., composition), which could be used to reconstruct the costly embedding  $v_x$  given the basis vectors. The rest of this section discusses each step mentioned above in detail.

**4.2.1 Step 1 - Noisy Fixed Cluster Generation and Assignment.** For a given attribute  $a$  (e.g., products in a shopping transaction stream), METEOR defines a set of clusters  $C^a = \{C_1^a, C_2^a, \dots, C_{|C|}^a\}$  and a cluster assignment function  $g^a : x \rightarrow c^x$  (where  $c^x \in C^a$ ), which assigns each unit  $x$  (e.g., white bread or skim milk if the attribute is a product in a shopping transaction stream) in  $a$  to a single cluster in  $C^a$ .  $C^a$  and  $g^a$  can be determined using either (1) explicitly available domain knowledge or (2) implicitly using a pretrained embedding space.

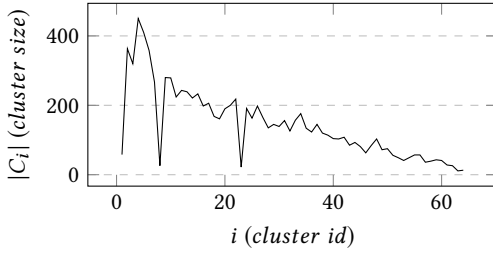
**(1) Explicit clusters.** These are the clusters generated using an explicitly available grouping scheme. As an example, products (i.e., a modality of a shopping transaction stream) can be grouped using explicitly available product categories. Both  $C^a$  and  $g^a$  can be determined using such a grouping, which do not generally change over time.

**(2) Implicit clusters.** The clusters in  $C^a$  can be generated using a clustering algorithm like KMeans [11] by clustering a pretrained embedding space (by optimizing Equation 3 using a subset  $R_{pre}$

<sup>4</sup>Sampling-based approaches require historical records to be stored, and the samples from the historical records are fed along with the recent records to update the embeddings incrementally.



**Figure 4: A smaller fraction of users in the TF dataset, which consists of 9238 users in total, change their clusters over different updating windows ( $\Delta W = 1$  day) if cluster changes are allowed (without Assumption 1)**



**Figure 5: The sizes of the noisy fixed clusters (64 clusters) generated for users in TF Dataset are uneven**

of  $R$ ) of  $a$ . Then  $g^a$  is defined such that it assigns each unit in the embedding space to the closest cluster based on the Euclidean distance to each cluster.

**Assumption 1.** METEOR assumes that the units do not change their clusters once they have been assigned to a noisy fixed cluster. This assumption is valid for explicit clusters. However, it may not be true for implicit clusters, as they are generated using the embeddings of the units, which are updated over time. As shown by [6], allowing hard cluster assignments to change over time could degrade the embedding space due to the sudden changes of the embeddings, which are incrementally learned over time. To preserve the validity of this assumption for the implicit clusters, we set the number of noisy fixed clusters to a small number (i.e.,  $< 1\%$  of the total number of units) to make the size of clusters big enough to minimize cluster changes over time. We empirically observe that Assumption 1 restricts only a few changes in the cluster assignments under the aforementioned setting, as shown in Figure 4. In Figure 4, the users in the TF dataset are clustered into 64 clusters based on their embeddings using KMeans (with the initialization of cluster centres using the cluster centres of the previous updating window) at the end of each updating window, and we observe that at most 0.06% users change their clusters over time.

**4.2.2 Step2 - Basis Vector Assignment.** For each noisy fixed cluster  $C_i^a$  of attribute  $a$ , a set of  $d$ -dimensional vectors  $B_i^a$  are assigned to represent  $C_i^a$ 's basis. Let  $|B_i^a|$  be the number of basis vectors in  $B_i^a$ , then  $B_i^a \in \mathbb{R}^{d \times |B_i^a|}$  (where  $d \gg |B_i^a|$ ).

**Number of basis vectors per cluster.** We empirically observe that the noisy fixed clusters generated using either explicit or implicit approaches are uneven in size, as shown in Figure 5. Hence,

---

**Algorithm 1: METEOR Learning**

---

**Input:** The noisy fixed cluster assignments  $C$   
The current compressed embeddings  $\hat{V}$   
The current basis vectors  $B$   
A collection of new records  $R_\Delta$

**Output:** The updated  $\hat{V}$  and  $B$

```

1  $V \leftarrow \emptyset$ ;
2 for unit  $x$  in  $R_\Delta$  do
3    $v_x \leftarrow \begin{cases} \text{initialize randomly} & \text{if } x \text{ is a new unit} \\ \text{compute using Eq. 8} & \text{otherwise} \end{cases}$ 
4    $V \leftarrow \{v_x\} \cup V$ 
5 end
   // Optimize  $L_{recon}$  using  $R_\Delta$ 
6 for epoch from 1 to  $N$  do
7   for  $r$  in  $R_\Delta$  do
8     | Update  $V$  to recover  $r$ 's attribute using Eq. 3;
9   end
10 end
   // Assign noisy fixed clusters for new units
11 for  $v_x \in V$  do
12   if  $x$  is a new unit then
13     | Assign  $v_x$  to the closest noisy fixed cluster
14   end
15 end
   // Optimize  $L_{comp}$  using  $V$ 
16 for  $v_x \in V$  do
17   | Update  $\hat{V}$  and  $B$  using Eq. 10;
18 end
19 Delete  $V$ ;
20 Return  $\hat{V}$  and  $B$ 

```

---

when assigning the basis vectors for different noisy fixed clusters, METEOR assigns more basis vectors to the large clusters using Equation 7 (based on the cluster sizes at the initialization) to accommodate uniform encoding for all the units. Let  $K^a$  be the total number of basis vectors assigned to different noisy fixed clusters of attribute  $a$ , then the number of basis vectors for  $C_i^a$ ,  $|B_i^a|$ , is calculated as,

$$|B_i^a| = \text{ceil}(K^a * \frac{|C_i^a|}{\sum_j |C_j^a|}) \quad (7)$$

where  $\text{ceil}(\cdot)$  is the standard ceiling operation.

**4.2.3 Step 3- Learning Compressed Embeddings and Basis Vectors.** For each unit  $x \in C_i^a$ , METEOR learns  $x$ 's costly embedding  $v_x$  as a linear combination of  $B_i^a$ ,

$$v_x = B_i^a \cdot \hat{v}_x \quad (8)$$

where  $\hat{v}_x$  is the memory-efficient compressed embedding of  $x$ ,  $\hat{v}_x \in \mathbb{R}^{|B_i^a|}$  and  $|B_i^a| \ll d$ . A trivial way to learn the memory-efficient embeddings  $\hat{V}$  and basis vectors  $B$  is by replacing the embeddings in Equation 3 using Equation 8. However, that approach is computationally expensive to perform in an online fashion as it introduces many matrix multiplications into the loss function.

Thus, METEOR decomposes the original loss function as follows,

$$L = L_{recon} + L_{comp} \quad (9)$$

where  $L_{comp}$  for a given unit  $x$  of attribute type  $a$  is defined as,

$$L_{comp} = (B_i^a \cdot \hat{v}_x - v_x)^2 + \lambda * ||\hat{v}_x|| \quad (10)$$

where  $\lambda$  is the weight given to the L1 regularization term. METEOR imposes L1 (i.e., Lasso) regularization on  $\hat{V}$  to make the memory-efficient representations  $\hat{V}$  as sparse as possible. This yields memory-efficient embeddings with a small fraction of non-zero values, which further reduces the memory requirement to store  $\hat{V}$  using sparse matrix storage formats.

**Initialization of Compressed Embeddings and Basis Vectors.** METEOR exploits the costly embedding in a pretrained embedding space,<sup>5</sup> which is used to generate noisy fixed clusters, and iteratively optimizes  $L_{comp}$  in Equation 10 using Adagrad to initialize  $\hat{V}$  and  $B$ .

**Incremental Learning of Compressed Embeddings and Basis vectors.** Then the compressed embeddings and the basis vectors are learned incrementally using the newly arrived records from the stream to incorporate recent information into the embeddings. For a given new set of records  $R_\Delta$ , METEOR adopts a sequential approach as shown in Algorithm 1 to learn the model, instead of jointly optimizing  $L$  in Equation 9. Initially, METEOR produces the costly embeddings for the units (using Equation 8) that appear in  $R_\Delta$  using the compressed embeddings and basis vectors returned at the end of the previous updating window (Line 2-5 in Algo 1). For new units, METEOR randomly initializes their costly embeddings. Then the costly embeddings are updated using Equation 3 to reconstruct the recent records. Then the new units of attribute type  $a$  are assigned to the noisy fixed clusters using the cluster assignment function  $g^a$ , introduced in Section 4.2.1 (Line 11-14). Subsequently,  $\hat{V}$  and  $B$  are updated using the recently updated costly embeddings by minimizing the compression loss in Equation 10 (Line 16-18).

Within an updating window, METEOR maintains costly embeddings only for the units appearing within the particular window, which is a small fraction compared to the total number of units (see Figure 2). At the end of the updating window, METEOR deletes the costly embeddings and only retains the compressed embeddings and basis vectors in memory, which requires considerably less memory (discussed in detail in Section 4.4). Thus, METEOR can be deployed on low-memory devices to learn recency-aware representations for different domains.

### 4.3 METEOR-PARALLEL

The other main challenge for a system like METEOR is scalability to high-speed data streams. The number of records processed by a system like METEOR should be at least similar to the rate of the stream to accommodate all the records in the stream. Since the rates of multi-modal streams can change significantly with the domain, a domain-agnostic model like METEOR should be able to adapt to a wide range of data rates. Thus, this section proposes a way to accelerate the learning process of METEOR using a parallel processing architecture with different memory domains. We denote

<sup>5</sup>The pretraining can be performed in a large server with sufficient memory capacity, as it needs to be performed only once.

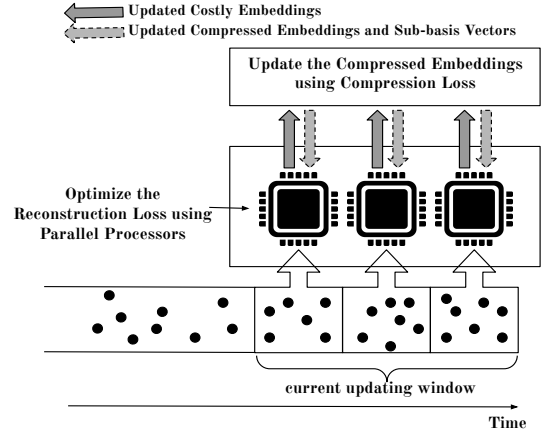


Figure 6: Architecture of METEOR-PARALLEL

this version of METEOR as "METEOR-PARALLEL" for the rest of this paper<sup>6</sup>.

As presented in the above section, METEOR optimizes the decomposed objective function in Equation 9 using two sequential steps: (1) update the costly embeddings to reconstruct the records in the current updating window; and (2) optimize the compression loss to update the compressed embeddings and the basis vectors. We have empirically identified that the latter step has a considerably lower time complexity compared to the former. Thus the bottleneck of the METEOR learning process lies with the former step. Also, we observed that learning Step 1 using parallel processors with separate memory units (e.g., a cluster of computers) does not deteriorate the quality of the compressed embeddings, and preserve the performance for downstream applications (see Figure 8). Hence, the architecture of METEOR-PARALLEL consists of multiple parallel processes to perform the first step and a single processor to perform the latter as shown in Figure 6.

Then for a given set of records in  $R_\Delta$ , METEOR-PARALLEL divides the records into  $p$  number of parallel processes. Each processor reads the most recent compressed embeddings and basis vectors to their own memory. Then each processor updates the costly embeddings (generated from the compressed embeddings and the basis vectors) to reconstruct the records arriving to each processor. Subsequently, all the updated costly embeddings are passed to the next stage, which centrally optimizes the compression loss to update the compressed embeddings and the basis vectors.

### 4.4 Complexity of METEOR

In this section, we analyse the complexity of METEOR and METEOR-PARALLEL using a multi-modal stream, in which each attribute  $a$  has  $|A^a|$  distinct units.

<sup>6</sup>The architecture of METEOR-PARALLEL is different from the conventional multi-threading architectures, which mostly use shared memory. Inside of a process in METEOR-PARALLEL, conventional multi-threading can be still applied for further acceleration. However, we emulate the architecture of METEOR-PARALLEL using multiple threads in this paper. We leave the performance of METEOR-PARALLEL on a real computer cluster as future work.

**4.4.1 Memory Complexity.** Assume that the number of noisy fixed clusters is  $|C^a| (\approx 1\% \times |A^a|)$  and the total number of basis vectors shared between noisy fixed clusters of attribute  $a$  is  $K^a$ . The space complexity to store conventional costly embeddings is  $O(d \times (\sum_a |A^a|))$ . In contrast, the space complexity of METEOR is  $O(\sum_a K^a (|A^a|/|C^a| + d))$  in the average case, where  $K^a/|C^a| \ll d$  and  $K^a \ll |A^a|$ . The memory complexity per processor in METEOR-PARALLEL remains the same.

**4.4.2 Time Complexity.** The time complexity of METEOR consists of: (1)  $O(ENM^2d \max(R_\Delta))$  to optimize the reconstruction loss (Step 1); and (2)  $O(Ed \sum_a \max(|a_\Delta|) K^a/|C^a|)$  to optimize the compression loss (Step 2), where  $E, N$ , and  $M$  are the number of epochs, the number of negative samples, and the maximum number of attributes of a record, respectively.  $|a_\Delta|$  is the distinct units of attributes type  $a$  that appear in  $R_\Delta$ , which is a smaller fraction of  $|A^a|$  as shown in Figure 2. The complexity of METEOR-PARALLEL with  $p$  processes remains same as for Step 2, and  $O(ENM^2d \max(R_\Delta)/p)$  for Step 1.

## 5 EXPERIMENTAL SETUP

**Datasets.** We conduct our experiments using three shopping transaction datasets:

- Complete Journey Dataset<sup>7</sup> (CJ) contains transactions at a retailer by 2,500 frequent shoppers over two years.
- Ta-Feng Dataset<sup>8</sup> (TF) includes shopping transactions of the Ta-Feng supermarket from November 2000 to February 2001.
- InstaCart Dataset<sup>9</sup> (IC) contains the shopping transactions of Instacart, an online grocery shopping centre, in 2017.

and three geo-tagged Twitter datasets collected from three urbanized cities:

- LA Dataset<sup>10</sup> (LA) contains around 1.2 million geo-tagged tweets from Los Angeles during the last quarter of 2014.
- NY Dataset<sup>10</sup> (NY) includes 1.5 million geo-tagged tweets collected from New York during the last quarter of 2014.
- MB Dataset<sup>10</sup> (MB) includes 263,363 geo-tagged tweets collected from Melbourne during the period from 2016 November to 2018 January.

The descriptive statistics of the datasets are shown in Table 1. The evaluation based on the datasets from different domains supports the domain-agnostic behaviour of the proposed model. As can be seen, the datasets from the same domain also have significantly different statistics. For example, TF has a shorter collection period and IC has a larger user base. This helps to evaluate the performance of METEOR in different environment settings.

**Baselines.** We compare METEOR with the following methods:

- DIM\_REDUCED reduces the size of the costly embedding vectors ( $d$ ) and learns the embeddings as shown in Section 4.1.
- $m$ -BIT QUANTIZATION evenly quantizes the values in the costly embeddings into  $2^m$ -bins as post processing.

**Table 1: Descriptive statistics of the datasets**

Shopping Transaction Datasets	# Transactions	# Users	# Items	# Baskets
Complete Journey (CJ)	2,595,733	2,500	92,339	276,483
Ta-Feng (TF)	464,118	9,238	7,973	77,202
Instacart (IC)	33,819,306	206,209	49688	3,421,083
Geo-tagged Twitter Datasets	# Tweets	# Users	# Locations	# Words
Los Angeles (LA)	1,188,405	153,626	18,271	353,586
New York (NY)	1,500,000	176,597	19,108	419,923
Melbourne (MB)	263,363	27,056	1,101	98,698

- HASH TRICK adopts modulo-division hashing to assign low-level units to clusters and a shared  $d$ -dimensional embedding is trained for each cluster. The divisor of the hashing function for an attribute  $a$  is set as  $\gamma \times A$ , where  $A$  is the total number of distinct units in  $a$ , and  $\gamma$  defines the value of the divisor as a proportion of  $A$ .
- DCN+HARD CLUSTERING adopts the deep clustering approach proposed in [25] to learn the costly embeddings. At the end of each updating window, the embedding space related to an attribute  $a$  is clustered into  $\gamma \times A$  clusters and the embeddings of the units are replaced by the corresponding cluster centers.

We compare a few online learning techniques with the proposed costly embedding learning approach (METEOR-FULL) in METEOR:

- METEOR-DECAY and METEOR-INFO adopt SGD optimization with the sampling-based online learning methods proposed in [27] and [20], respectively.
- METEOR-CONS adopts SGD optimization with the constraint-based online learning approach proposed in [27].

**Parameter Settings.** The two main parameters of METEOR are  $|C^a|$  and  $K^a$ . For most of the experiments, we set  $|C^a| = 1\%|A^a|$  and  $K^a = 10\% * |A^a|$  ( $|A^a|$  is the total distinct units of attribute  $a$  in the pretrained embedding space), otherwise the parameter values are specified. We present the results with different  $|C^a|$  and  $K^a$  values in Figure 7. The weight given to the sparsity constraint  $\lambda$  is set to 0.001 in METEOR after performing a grid search over  $[0, 0.0001, 0.001, 0.01, 0.1, 1]$  values.

In addition, all the aforementioned techniques share three common parameters (default values are given in brackets): (1) the costly embedding dimension  $d$  (300), (2) the SGD learning rate  $\eta$  (0.05), (3) the negative samples  $|N_x|$  (3), and (4) the number of epochs  $N$  (50). We set  $\tau = 0.1$  for sampling based online learning techniques.

**Evaluation Metrics.** METEOR is quantitatively evaluated using two retrieval tasks: (1) *intra-basket item retrieval task* for shopping transaction datasets; and (2) *location retrieval task* for geo-tagged Twitter datasets. Both tasks follow a similar experimental setup. Similar to previous work [27], we adopt the following procedure to evaluate the performance of each retrieval task. For each record in the test set, we select one unit (e.g., a product for *intra-basket item retrieval* or the location for *location retrieval*) as the target prediction and the rest of the units of the record as the context. We mix the ground truth target unit with a set of  $M$  negative samples (i.e., a set of units that have the type of the ground truth) to generate a candidate pool to rank.  $M$  is set to 10 for all the experiments. Then the size- $(M + 1)$  candidate pool is sorted to get the rank of the ground truth. The average similarity of each candidate unit to the context of the corresponding test instance is used to

<sup>7</sup><https://www.dunnhumby.com/careers/engineering/sourcefiles>

<sup>8</sup><http://www.bigdatalab.ac.cn/benchmark/bm/dd?data=Ta-Feng>

<sup>9</sup><https://www.instacart.com/datasets/grocery-shopping-2017>

<sup>10</sup><https://drive.google.com/file/d/0Byzh4bOatCRHdmRVZ1YVZqSZA/view>

**Table 2: Results for *intra-basket item retrieval***

method	parameter(s) for reducing model size	Additional memory for a new unit	CJ Dataset			TF Dataset			IC Dataset		
			Model size	MRR	R@1	Model size	MRR	R@1	Model size	MRR	R@1
METEOR-FULL	costly embeddings (d=300)	<b>2.34KB</b>	<b>217MB</b>	<b>0.6013</b>	<b>0.4325</b>	<b>39MB</b>	<b>0.5166</b>	<b>0.3466</b>	<b>586MB</b>	<b>0.7478</b>	<b>0.5859</b>
METEOR-INFO	costly embeddings (d=300)	2.34KB	217MB	0.5991	0.4275	39MB	0.4046	0.2205	586MB	0.7482	0.5852
METEOR-DECAY	costly embeddings (d=300)	2.34KB	217MB	0.5984	0.4221	39MB	0.402	0.2186	586MB	0.7117	0.5442
METEOR-CONS	costly embeddings (d=300)	2.34KB	217MB	0.4610	0.2742	39MB	0.3996	0.2031	586MB	0.5942	0.4193
METEOR	$ C^a  = 0.5\% A^a , K^a = 10\% A^a $	<b>0.16KB</b>	<b>78MB</b>	<b>0.5974</b>	<b>0.4293</b>	<b>14MB</b>	<b>0.5013</b>	<b>0.3287</b>	<b>112MB</b>	<b>0.7203</b>	<b>0.5507</b>
	$ C^a  = 1\% A^a , K^a = 10\% A^a $	<b>0.08KB</b>	<b>48MB</b>	<b>0.587</b>	<b>0.4098</b>	<b>9MB</b>	<b>0.4961</b>	<b>0.3203</b>	<b>71MB</b>	<b>0.7084</b>	<b>0.5411</b>
DIM_REDUCT	d = 100	0.76KB	72MB	0.5773	0.4044	13MB	0.4762	0.2940	195MB	0.6923	0.5213
	d = 50	0.38KB	36MB	0.5495	0.3664	7MB	0.4509	0.2713	98MB	0.6628	0.4825
	d = 25	0.19KB	18MB	0.5178	0.3323	4MB	0.4228	0.2305	49MB	0.6284	0.4381
QUANTIZATION	8 bit quant.	0.59KB	54MB	0.5677	0.3722	10MB	0.4672	0.2892	147MB	0.6836	0.5077
	4 bit quant.	0.29KB	27MB	0.5453	0.3502	5MB	0.4478	0.2700	73MB	0.6447	0.4603
	2 bit quant.	0.15KB	14MB	0.4321	0.2801	3MB	0.3217	0.1413	37MB	0.5573	0.3705
HASH TRICK	$\gamma = 30\%$	4B	65MB	0.5376	0.3487	12MB	0.4335	0.2535	176MB	0.6558	0.4727
	$\gamma = 20\%$	4B	43MB	0.4993	0.3106	8MB	0.3711	0.1987	117MB	0.6248	0.4376
	$\gamma = 10\%$	4B	22MB	0.4677	0.2988	4MB	0.3417	0.1786	58.6MB	0.5688	0.3848
DCN + HARD CLUSTERING	$\gamma = 30\%$	4B	65MB	0.5477	0.3588	12MB	0.4577	0.2724	176MB	0.6731	0.4906
	$\gamma = 20\%$	4B	43MB	0.5321	0.3411	8MB	0.4122	0.2236	117MB	0.6482	0.4583
	$\gamma = 10\%$	4B	22MB	0.4882	0.3075	4MB	0.3876	0.2033	58.6MB	0.5883	0.4017

produce the ranking of the candidate pool. Cosine similarity is used as the similarity measure for all the baselines.

If the model is well trained, then higher ranked units are most likely to be the ground truth. Hence, we use two different evaluation metrics to analyze the ranking performance:

- (1) Mean Reciprocal Rank (MRR) =  $\frac{\sum_{q=1}^Q 1/rank_i}{|Q|}$
- (2) Recall@k (R@k) =  $\frac{\sum_{q=1}^Q \min(1, \lfloor k/rank_i \rfloor)}{|Q|}$

where  $Q$  is the set of test queries and  $rank_i$  refers the rank of the ground truth label for the  $i$ -th query.  $\lfloor \cdot \rfloor$  is the floor operation. A good ranking performance should yield higher values for the both evaluation metrics.

We divide the records in each data stream into different updating windows such that each window has a length of  $\Delta w$  ( $\Delta w = 1$  day for shopping transaction datasets and  $\Delta w = 1$  hour for geo-tagged Twitter datasets). The first half of the period for each dataset is used to pretrain costly embeddings, which are subsequently used to produce noisy-fixed clusters. We randomly select 20 query updating windows from the second half of the period for each dataset, and all the records in the randomly selected time windows are used as test instances. For each query window, we only use the records that arrive before the query window to train different models. We ignore timestamps in both types of streams in the embedding as they do not substantially affect the performance of the tasks. The locations in geo-tagged Twitter streams are discretized into  $300m \times 300m$  small grids to make them feasible for embedding.

## 6 RESULTS

**(1) Compressed Embedding Learning in METEOR.** Table 2 and Table 3 show the results collected for *intra-basket item retrieval* and *location retrieval* respectively. METEOR shows significantly better results than the comparable baseline models, which are the models with similar size. For example, if we consider the results collected for *intra-basket retrieval* using the IC dataset, the model size of METEOR at ( $K^a = 10\%|A^a|, |C^a| = 1\%|A^a|$ ) is similar to: DIM REDUCT at ( $d = 25$ ); QUANTIZATION at (4 bit quant.); HASH

TRICK at ( $\gamma = 10\%$ ); and DCN+HARD CLUSTERING at ( $\gamma = 10\%$ ). Compared to the comparable models, METEOR outperforms the best baseline by 9.88% in MRR and 17.55% in *Recall@1*. This observation is consistent for both *intra-basket item retrieval* and *location retrieval*.

Out of the baselines, DIM REDUCT and QUANTIZATION are the strongest baselines, but they have linearly increasing model sizes with respect to the total number of distinct units. Thus, the additional memory required to represent a newly seen unit is higher for those two baselines. For example, QUANTIZATION at (4 bit quant.) requires 363% more memory per unit compared to the corresponding METEOR at ( $K^a = 10\%|A^a|, |C^a| = 1\%|A^a|$ ). The other two baselines: HASH TRICK; and DCN+HARD CLUSTERING, have significantly lower overheads for new units than METEOR. However, they perform a hard cluster assignment for each unit, which restricts the flexibility of representations, thus yielding poor performance for the downstream tasks. In DCN+HARD CLUSTERING, the hard clustering is performed incrementally at the end of each updating window. Subsequently, for a given unit, the embedding is taken as the centre of the cluster that the unit belongs to. However, as shown in [6], such hard assignment can degrade the embedding space, which could be the reason for having poor results with DCN+HARD CLUSTERING. HASH TRICK is the worst baseline, which could be due to the hard cluster assignment performed randomly without considering the semantics of the units. Hence, it is important to consider the semantics when learning memory efficient representations.

**Explicit Clusters as Noisy Fixed Clusters.** As we discussed in Section 4.2.1, METEOR can exploit the knowledge available in explicit grouping schemes when assigning the noisy fixed clusters for units. The results collected for *intra-basket item retrieval* using such explicit product categories in CJ datasets are shown in Table 4. As can be seen, the compressed embeddings in METEOR can even outperform METEOR-FULL while achieving 83% ( $= (217MB - 37MB) \times 100\% / 217MB$ ) reduction in memory compared to METEOR-FULL, when the noisy fixed clusters are assigned using the product category information. This could be due to the shared basis vectors inside noisy fixed clusters. Thus, they can capture the



**Table 3: Results for location retrieval**

method	parameter(s) for reducing model size	Additional memory for a new unit	LA Dataset			NY Dataset			MB Dataset		
			Model size	MRR	R@1	Model size	MRR	R@1	Model size	MRR	R@1
METEOR-FULL	costly embeddings (d=300)	2.34KB	1118MB	0.7997	0.6943	1313MB	0.7956	0.6894	274MB	0.6157	0.4476
METEOR-INFO	costly embeddings (d=300)	2.34KB	1118MB	0.8053	0.6943	1313MB	0.8027	0.6927	274MB	0.6021	0.4388
METEOR-DECAY	costly embeddings (d=300)	2.34KB	1118MB	0.7614	0.6488	1313MB	0.775	0.6590	274MB	0.5808	0.4169
METEOR-CONS	costly embeddings (d=300)	2.34KB	1118MB	0.7602	0.6488	1313MB	0.7701	0.6562	274MB	0.5762	0.4001
METEOR	$ C^a  = 0.5 A^a , K^a = 10 A^a $	0.16KB	204MB	0.7899	0.6858	241MB	0.7943	0.6805	62MB	0.6004	0.4366
	$ C^a  = 1 A^a , K^a = 10 A^a $	0.08KB	128MB	0.7701	0.6631	152MB	0.7803	0.6714	31MB	0.5897	0.4287
DIM_REDUCT	d = 100	0.76KB	373MB	0.7426	0.6134	438MB	0.7398	0.6188	91MB	0.5403	0.3614
	d = 50	0.38KB	187MB	0.7116	0.5683	219MB	0.7073	0.5692	46MB	0.5022	0.3245
	d = 25	0.19KB	93MB	0.6882	0.5385	109MB	0.6702	0.5173	23MB	0.4595	0.2806
QUANTIZATION	8 bit quant.	0.59KB	280MB	0.7262	0.5882	328MB	0.7288	0.5946	69MB	0.5333	0.3572
	4 bit quant.	0.29KB	140MB	0.6903	0.5483	164MB	0.6929	0.5493	34MB	0.4859	0.3006
	2 bit quant.	0.15KB	70MB	0.5883	0.4277	82MB	0.5892	0.4196	17MB	0.4007	0.2177
HASH TRICK	$\gamma = 30\%$	4B	335MB	0.7001	0.5481	394MB	0.7065	0.5582	82MB	0.5338	0.3520
	$\gamma = 20\%$	4B	224MB	0.6638	0.5083	263MB	0.6690	0.5173	55MB	0.4961	0.3169
	$\gamma = 10\%$	4B	112MB	0.6104	0.4854	131MB	0.6152	0.4502	27MB	0.4517	0.2752
DCN + HARD CLUSTERING	$\gamma = 30\%$	4B	335MB	0.7177	0.5764	394MB	0.7208	0.5872	82MB	0.5382	0.3520
	$\gamma = 20\%$	4B	224MB	0.6843	0.5375	263MB	0.6943	0.5493	55MB	0.5099	0.3287
	$\gamma = 10\%$	4B	112MB	0.6429	0.5104	131MB	0.6544	0.4921	27MB	0.4728	0.2970

**Table 4: Results for intra-basket item retrieval with explicit product clusters (categories 92,339 products in to 2384 product categories) in CJ Dataset as noisy fixed clusters**

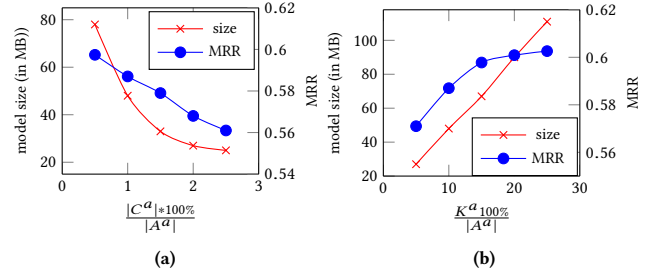
The variant of METEOR	Average memory for a new unit	Model size	MRR	R@1
Using costly embeddings (d=300)	2.34KB	217MB	0.6013	0.4325
Without explicit product clusters $ C^a  = 0.5 A^a , K^a = 10 A^a $ $ C^a  = 1 A^a , K^a = 10 A^a $	0.25KB	78MB	0.5974	0.4293
	0.13KB	48MB	0.587	0.4098
With explicit product clusters $ C^a  = 0.5 A^a , K^a = 10 A^a $ $ C^a  = 1 A^a , K^a = 10 A^a $	0.25KB	59MB	<b>0.6097</b>	<b>0.4401</b>
	0.13KB	37MB	0.6028	0.4337

**Table 5: Ablation Study of Elements in METEOR**

Method	CJ Dataset		LA Dataset	
	MRR	R@1	MRR	R@1
METEOR	0.587	0.4098	0.7701	0.6631
(-) weighted basis vector assignment	0.5703	0.3995	0.7577	0.6304
(-) sparsity constraint	0.5852	0.4098	0.7695	0.6631

additional knowledge introduced by explicit clusters too. This could help to improve the embeddings of rarely-occurring units (which have inaccurate embeddings in general) based on their neighbours in the same explicit cluster. To verify this point, the accuracy of the predicted labels for rarely occurring target test instances (appearing < 10 times) are examined. For those test instances, the compressed embeddings of METEOR with explicit noisy fixed clusters outperforms the costly embeddings of METEOR by nearly 2% with respect to MRR. Thus, we can conclude that the compressed embeddings of METEOR with explicit noisy fixed clusters predicts rarely occurring ground truth instances more accurately than the costly embeddings.

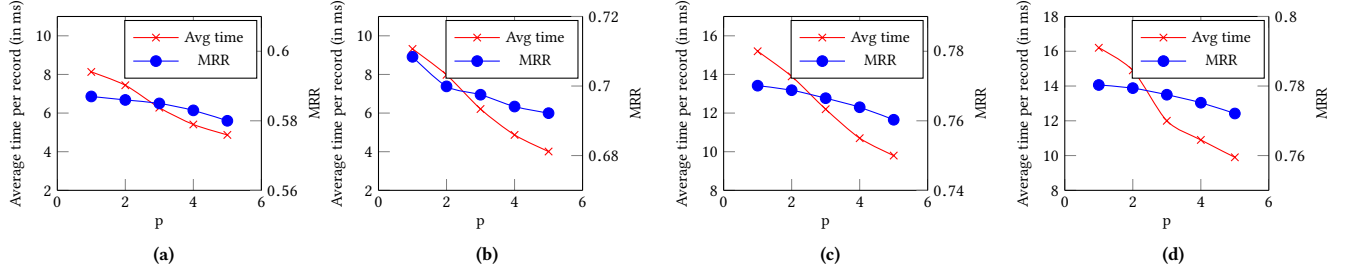
**Ablation Study.** To check the significance of the role of the each element in METEOR, we perform an ablation study as shown in Table 5: (1) by removing *weighted basis vector assignment*, which uniformly distributes basis vectors among noisy fixed clusters; and (2) by removing the *sparsity constraint* in Equation 10. As shown, the weighted basis vector assignment plays a significant role in METEOR. Although the sparsity constraint does not account for a significant performance improvement, we empirically observe

**Figure 7: Parameter Sensitivity on CJ: (a) MRRs for different  $|C^a|$  values with  $K^a = 10|A^a|$ ; and (b) MRRs for different  $K^a$  values with  $|C^a| = 1|A^a|$** 

that it yields compressed embeddings with many zeros, which could be exploited to reduce the memory to store the compressed embeddings using sparse matrix storage formats.

**Parameter Sensitivity.** In Figure 7, we check the performance of METEOR, with different  $K^a$  and  $|C^a|$  values. For a given fixed  $K^a$  value, when  $|C^a|$  increases (see Figure 7a), the number of basis vectors per cluster reduces. Thus, the number of dimensions for compressed embedding reduces, in turn reducing the model size. Meanwhile, small noisy fixed clusters (with high  $|C^a|$  values) increase the violations of Assumption 1 (see Section 4.2.1). This could be the reason for the performance drop when  $|C^a|$  increases. As shown in Figure 7b, when  $K^a$  increases, both model sizes and MRRs for retrieval tasks monotonically increase and ultimately reach the model size and the performance of METEOR-FULL.

**(2) Online Learning in METEOR.** Table 2 and Table 3 also show the performance of the proposed online learning approach of METEOR (i.e., METEOR-FULL) to update costly embeddings. Comparing METEOR-FULL with the other online learning variants of METEOR: METEOR-INFO; METEOR-DECAY; and METEOR-CONS, METEOR-FULL's results are comparable (mostly superior) with sampling-based online learning variants of METEOR (i.e., METEOR-DECAY and METEOR-INFO), which store historical records to avoid overfitting to recent records. Also, METEOR-FULL outperforms METEOR-CONS as much as 30% with respect to MRR, which has a



**Figure 8: Average time taken to process a record and MRR of METEOR-Parallel with  $p$  number of parallel processors: (a) for intra-basket item retrieval using CJ; (b) for intra-basket item retrieval using IC; (c) for location retrieval using LA; and (d) for location retrieval using NY**

similar memory complexity of METEOR-FULL. Hence, the proposed adaptive optimization-based online learning technique in METEOR achieves the performance of the state-of-the-art online learning methods (i.e., METEOR-DECAY and METEOR-INFO) in a memory-efficient manner without storing any historical records.

**(3) METEOR-PARALLEL.** Table 8 shows the results collected with METEOR-PARALLEL with different numbers of parallel processes. For both retrieval tasks, METEOR-PARALLEL yields around 50% reduction in the time taken to process a single record with 5 parallel processes. In contrast, the drop in the performance with respect to MRR with 5 processes are slight, which are 1.2%, 2.3%, 1.3%, and 1.1% for CJ, IC, LA, and NY respectively. Thus, METEOR-PARALLEL could be used to accelerate the embedding learning process of METEOR, while preserving the quality of the embeddings.

## 7 CONCLUSION

In this work, we proposed METEOR, which learns compact representations using multi-modal data streams in an online fashion. The learning of METEOR could be speeded up using parallel processes with different memory domains. Our results show that METEOR is a domain-agnostic framework, which can substantially reduce the memory complexity of conventional embedding learning approaches while preserving the quality of the embeddings.

METEOR achieves around 80% reduction in memory compared to the conventional costly embeddings without sacrificing performance by sharing parameters inside the semantically meaningful groupings of the multi-modal units. Hence, integrating METEOR with other similar explicit/implicit knowledge bases could be a promising research direction. Also, METEOR decides the number of shared parameters for each semantic group (i.e., noisy fixed clusters) based on their size. Besides, other factors could be considered (e.g., cluster shape) when assigning the basis vectors for noisy fixed clusters. In this work, we emulate the architecture of METEOR-PARALLEL using multi-threading, and we leave its actual implementation in a cluster of machines as future work. Also, we aim to explore the applications of METEOR for other domains.

## ACKNOWLEDGEMENT

This research was financially supported by Melbourne Graduate Research Scholarship and Rowden White Scholarship.

## REFERENCES

- [1] Anish Acharya, Rahul Goel, Angeliki Metallinou, and Inderjit Dhillon. 2019. Online embedding compression for text classification using low rank matrix factorization. In *Proc. of AAAI*.
- [2] Yoshua Bengio, Aaron Courville, and Pascal Vincent. 2013. Representation learning: A review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2013).
- [3] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2017. Enriching Word Vectors with Subword Information. *Transactions of the Association for Computational Linguistics* (2017).
- [4] Ting Chen, Martin Renqiang Min, and Yizhou Sun. 2018. Learning k-way d-dimensional discrete codes for compact embedding representations. *Proc. of ICML* (2018).
- [5] John Duchi, Elad Hazan, and Yoram Singer. 2010. Adaptive subgradient methods for online learning and stochastic optimization. *Proc. of COLT* (2010).
- [6] Xifeng Guo, Long Gao, Xinwang Liu, and Jianping Yin. 2017. Improved deep embedded clustering with local structure preservation. In *Proc. of IJCAI*.
- [7] Richard W Hamming. 1950. Error detecting and error correcting codes. *The Bell System Technical Journal* (1950).
- [8] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proc. of CVPR*.
- [9] David A Huffman. 1952. A method for the construction of minimum-redundancy codes. *Proc. of the IRE* (1952).
- [10] Yoon Kim, Yacine Jernite, David Sontag, and Alexander M Rush. 2016. Character-aware neural language models. In *Proc. of AAAI*.
- [11] Stuart Lloyd. 1982. Least squares quantization in PCM. *IEEE Transactions on Information Theory* (1982).
- [12] Avner May, Jian Zhang, Tri Dao, and Christopher Ré. 2019. On the downstream performance of compressed word embeddings. In *Proc. of NeurIPS*.
- [13] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Proc. of NIPS*.
- [14] Aliakbar Panahi, Seyran Saeedi, and Tom Arodz. 2019. word2ket: Space-efficient Word Embeddings inspired by Quantum Entanglement. *arXiv:1911.04975* (2019).
- [15] Jeffrey Pennington, Richard Socher, and Christopher D Manning. 2014. Glove: Global vectors for word representation. In *Proc. of EMNLP*.
- [16] Rico Sennrich, Barry Haddow, and Alexandra Birch. 2015. Neural machine translation of rare words with subword units. *arXiv:1508.07909* (2015).
- [17] Hao-Jun Michael Shi, Dheevatsa Mudigere, Maxim Naumov, and Jiyan Yang. 2019. Compositional Embeddings Using Complementary Partitions for Memory-Efficient Recommendation Systems. *arXiv:1909.02107* (2019).
- [18] Raphael Shu and Hideki Nakayama. 2017. Compressing word embeddings via deep compositional code learning. *arXiv:1711.01068* (2017).
- [19] Amila Silva, Shanika Karunasekera, Christopher Leckie, and Ling Luo. 2019. US-TAR: Online Multimodal Embedding for Modeling User-Guided Spatiotemporal Activity. *Proc. of IEEE BigData* (2019).
- [20] Amila Silva, Shanika Karunasekera, Christopher Leckie, and Ling Luo. 2019. US-TAR: Online Multimodal Embedding for Modeling User-Guided Spatiotemporal Activity. *Proc. of IEEE-BigData* (2019).
- [21] Karen Simonyan and Andrew Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv:1409.1556* (2014).
- [22] Jun Suzuki and Masaaki Nagata. 2016. Learning Compact Neural Word Embeddings by Parameter Space Sharing. In *Proc. of IJCAI*.
- [23] Dan Tito Svenstrup, Jonas Hansen, and Ole Winther. 2017. Hash embeddings for efficient word representations. In *Proc. of NIPS*.

- [24] Kilian Weinberger, Anirban Dasgupta, John Langford, Alex Smola, and Josh Attenberg. 2009. Feature hashing for large scale multitask learning. In *Proc. of ICML*.
- [25] Bo Yang, Xiao Fu, Nicholas D Sidiropoulos, and Mingyi Hong. 2017. Towards k-means-friendly spaces: Simultaneous deep learning and clustering. In *Proc. of ICML*.
- [26] Zhilin Yang, William W Cohen, and Ruslan Salakhutdinov. 2016. Revisiting semi-supervised learning with graph embeddings. *arXiv:1603.08861* (2016).
- [27] Chao Zhang, Keyang Zhang, Quan Yuan, Fangbo Tao, Luming Zhang, Tim Hanratty, and Jiawei Han. 2017. React: Online multimodal embedding for recency-aware spatiotemporal activity modeling. In *Proc. of SIGIR*.
- [28] Daokun Zhang, Jie Yin, Xingquan Zhu, and Chengqi Zhang. 2018. Network representation learning: A survey. *IEEE Transactions on Big Data* (2018).