METRIC LEARNING USING DEEP RECURRENT NETWORKS FOR VISUAL CLUSTERING AND RETRIEVAL

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

OĞUL CAN

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
ELECTRICAL AND ELECTRONICS ENGINEERING

SEPTEMBER, 2018

Approval of the thesis:

METRIC LEARNING USING DEEP RECURRENT NETWORKS FOR VISUAL CLUSTERING AND RETRIEVAL

submitted by **OĞUL CAN** in partial fulfillment of the requirements for the degree of **Master of Science in Electrical and Electronics Engineering Department, Middle East Technical University** by,

ABSTRACT

METRIC LEARNING USING DEEP RECURRENT NETWORKS FOR VISUAL CLUSTERING AND RETRIEVAL

Can, Oğul
M.S., Department of Electrical and Electronics Engineering
Supervisor : Prof. Dr. A. Aydın Alatan

September, 2018, 92 pages

Learning an image similarity metric plays a key role in visual analysis, especially for the cases where a training set contains a large number of hard negative samples that are difficult to distinguish from other classes. Due to the outstanding results of the deep metric learning on visual tasks, such as image clustering and retrieval, selecting a proper loss function and a sampling method becomes a central issue to boost the performance. The existing metric learning approaches have two significant drawbacks; inadequate mini-batch sampling and disregarding higher-order relations between data samples. In this thesis, two novel methods are proposed to alleviate these deficiencies. At first, a novel loss function is introduced to identify multiple similar examples in a local neighborhood. Moreover, a novel batch construction method is presented to select representative hard negatives. The training of a deep network is achieved by using this novel cost function through the proposed batch construction approach. In order to consider higher-order relations between samples, a novel deep metric learning framework that contains recurrent neural networks architecture is proposed. Extensive experimental results on three publicly available datasets show that proposed approaches yield competitive or better performance in comparison with state-of-the-art metric learning methods.

Keywords: Deep Metric Learning, Recurrent Neural Networks, Local Neighborhood Sampling

ÖZ

GÖRSEL KÜMELEME VE BULGETİR İÇİN DERİN YİNELGELİ AĞLAR KULLANARAK METRİK ÖĞRENME

Can, Oğul
Yüksek Lisans, Elektrik ve Elektronik Mühendisliği Bölümü
Tez Yöneticisi : Prof. Dr. A. Aydın Alatan

2018 , 92 sayfa

Görüntü benzerlik metriğini öğrenmek, özellikle diğer sınıflardan zor ayrışan olumsuz örneklerin çokça barındığı eğitim setlerinin görüntü analizlerinde anahtar rol oynamaktadır. Derin metrik öğrenmenin görsel kümeleme ve bulgetir gibi çalışma alanlarındaki başarısı nedeniyle uygun kayıp fonksiyonu ve örnekleme yöntemini seçmek, başarımı arttırmak için odak noktası haline gelmektedir. Ancak, varolan metrik öğrenme yaklaşımlarının iki kayda değer sınırlaması mevcuttur; yetersiz küçük yığın örnekleme yöntemleri kullanılması ve verideki üst seviye ilişkileri gözardı etmeleridir. Bu tezde bahsedilen eksiklikleri gidermek için yenilikçi iki yöntem önerilmektedir. İlk olarak, yerel komşuluktaki çoklu benzer örnekleri tanımlamak için yeni bir kayıp fonksiyonu tanıtılmaktadır. Devamında, diğer sınıflardan zor ayrışan temsili olumsuz örnekleri seçmek için yenilikçi bir olumsuz yığın oluşturma yöntemi sunulmuştur. Derin ağların eğitilmesi önerilen yığın oluşturma yöntemi kullanılarak yenilikçi kayıp fonksiyonuyla sağlanmıştır. Üst seviye ilintileri modellemek için yenilikçi bir yinelgeli ağ mimarisi kullanan derin metrik öğrenme çerçevesi önerilmiştir. Üç kamusal veri setinden alınan kapsamlı deneysel sonuçlar sonunda, önerilen yaklaşımların önde gelen metrik öğrenme yöntemleriyle kıyaslandığında rekabetçi ya da daha iyi başarım sağladığı gösterilmiştir.

Anahtar Kelimeler: Derin Metrik Öğrenme, Derin Yinelgeli Ağlar, Yerel Komşuluk Örnekleme

To Canan and Canan Can...

ACKNOWLEDGMENTS

First and foremost, I would like to express my deepest gratitude to my supervisor Prof. Dr. A. Aydın Alatan for all of his contributions to my academic endeavors. He is the man who changed my interests in the first place as I was so fascinated by his wisdom and intelligence. During my time at METU Center for Image Analysis (OGAM), he has always shown me endless tolerance and patience for my shortcomings. I want to thank him for trusting me even at the times when I was ready to give up on myself. I am truly indebted to him for providing me insightful comments and scholarly feedbacks throughout my journey, which eventually enabled this thesis to came into existence. I feel privileged to have this brilliant person as a mentor and be a part of his research team.

Furthermore, I would like to extend my gratitude to the current and past members of OGAM. I offer my special thanks to Yeti Ziya Gürbüz who has not only been one of my biggest supporters but also a great friend and a genius counselor. His excellent background knowledge and our academic discussions helped me look deeper into my findings and enrich my interpretations. I thank Ece Selin Böncü for being my gym buddy and partner in crime. I also thank one of her many personalities, sweet potato, for being a lovely friend. I offer my sincerest thanks to Ömürcan Kumtepe, for he was the one whose companionship enabled me to adapt to my new workplace. I thank Dr. Alpez Koz for sharing his wisdom and wise guidance in both academic and personal life. I thank the talented artist of our laboratory, Emre Can Kaya, for showing me how to look at academic studies from a creative perspective. I thank İzlen Geneci and Aybüke Erol for making my time in the laboratory enjoyable and being good friends. I also thank Esat Kalfaoğlu for permitting me to use his computer during this thesis work and his nice personality. I also would like to acknowledge Bahar Taşkesen, Emrecan Batı, Berkay Yeşilyurt, Dr. Gökhan Koray Gültekin, Mustafa Kütük, Mertalp Öcal, İhsan Emre Üstün, Mustafa Ergül and Alp Eren Sarı as great colleagues.

Most importantly, I would like to offer a heartful of thanks to Büşra Temel for her endless love and support. Without her by my side, I would lose my mind and could not survived. She has always been able to calm me down and put me back on my feet whenever I felt overwhelmed by the huge amount of work required to finish this thesis. On the other hand, although they hindered me throughout this painful journey, I would also like to thank my best friends Ömercan Bayıraş, Selin Özkul, Utku Çalış and Yılmaz Oğuz Ölke for growing up with me (the list is alphabetically ordered).

My sincerest thanks go to my parents, Canan Can and Mehmet Canan Can. I want to thank my mother for years of support and lifetime funding. I could not possibly thank her enough since she still takes care of me as if I were a baby. I thank my father for making me believe that I can achieve anything that my heart desires. He shaped the man I am today. Moreover, I thank my lovely sister Anıl Didem Erdoğan for allowing me to be raised as a single child. Finally, I thank my aunt Hülya Kolunan and my uncle Gencer Kolunan for believing in me and encouraging me to pursue an academic career.

Last but not least, I also would like to take this chance to thank ASELSAN for funding my research project.

TABLE OF CONTENTS

LIST OF TABLES

TABLES

LIST OF FIGURES

FIGURES

LIST OF ABBREVIATIONS

LMNN	Large Margin Nearest Neighbors
ITML	Information-Theoretic Metric Learning
CNN	Convolutional Neural Network
NMI	Normalized Mutual Information
RNN	Recurrent Neural Network
ANN	Artificial Neural Network
MLP	Multilayer Perceptron
KL	Kullback-Leibler
GPU	Graphics Processing Unit
Adam	Adaptive Moment Estimation
BPTT	Backpropagation Through Time
ReLu	Rectified Linear Unit
GRU	Gated Recurrent Units
LSTM	Long Short-term Memories
DAML	Deep Adversarial Metric Learning

CHAPTER 1

INTRODUCTION

Metric learning aims to learn a feature mapping such that similar examples are mapped to closer points on an embedding space and dissimilar examples are mapped to relatively far away points. Learning a distance metric to measure similarities between image pairs have been widely used in a variety of visual tasks, such as image clustering [9], person re-identification [10, 11, 12], face identification [13, 14], feature based retrieval [1, 15], challenging classification problems [16, 17, 18],visual tracking [19, 20], etc.

Existing metric learning methods are mostly divided into two main groups as linear and non-linear approaches [21]. While linear methods aim to learn a metric that is formulated by a linear relation such as *Mahalanobis* distance [13, 22, 23], non-linear approaches exploit kernels and deep neural networks to model non-linear relations in the data. Following the superior achievement of deep learning in numerous visual tasks [24, 25, 26, 27], metric learning methods based on deep learning are proposed to learn non-linear mappings [1, 9, 15, 28, 29, 30, 31, 32].

Although these methods yield promising results, they suffer from two significant limitations. Firstly, due to the inefficient mini-batch construction methods to select *hard negatives* which are difficult samples to classify from other classes, they keep dissimilar samples away from similar examples by considering only a pair of samples which belong to the same class. Secondly, existing metric learning frameworks are trained with respect to some loss functions that are optimized by utilizing lower-order relations in the data. This approach might lead to sub-optimal convergence in an higher-order solution space [31].

In order to address the above-mentioned issues, two associated methods are proposed in the scope of this thesis. At first, a novel loss function is proposed to utilize multiple similar examples, while dragging multiple dissimilar samples away from them. Then, a novel local neighborhood sampling approach is introduced in order to train proposed loss function by inspiring from *hard negative mining* [33] and *self-paced learning* [34]. Finally, in the light of these introduced loss function and sampling method, a novel deep metric learning framework that utilizes RNN architectures is proposed in order to capture higher-order relations between samples.

1.1 Problem Definition

Assume $\boldsymbol{x}_i$ denotes a sample in a training set and $\boldsymbol{x}_j^+ \in \mathbb{P}_i$ represents each sample in the positive set that is constructed from examples that belong to same class as sample i whereas $\boldsymbol{x}_k^- \in \mathbb{N}_i$ indicates each dissimilar example to the sample i in the negative set. The goal of the metric learning is to learn a mapping, $f_{\boldsymbol{\theta}}(.)$, where $\boldsymbol{\theta}$ denote the trainable parameters, so that the distance between $f_{\boldsymbol{\theta}}(\boldsymbol{x}_i)$ and $f_{\boldsymbol{\theta}}(\boldsymbol{x}_j^+)$ is kept smaller while each distance from $f_{\boldsymbol{\theta}}(\boldsymbol{x}_k^-)$ to $f_{\boldsymbol{\theta}}(\boldsymbol{x}_i)$ becomes larger.

An example case is visualized in Figure 1.1. Before utilizing metric learning, the distances between pairs of the similar and dissimilar examples which are indicated in green and red colors, respectively, are assumed to be equal. After applying mapping into the learned embedding space, for instance, in Figure 1.1, embedding representations of two bicycles are getting closer to each other whereas an apple and a mobile phone is getting far away.

Clustering and retrieval tasks are widely utilized to evaluate metric learning methods in the literature [1]. Clustering is an *unsupervised* problem of grouping similar examples into the same groups (*clusters*), while the unsupervised task indicates that this process does not exploit the target distribution of inputs with labels to solve some machine learning problems [35]. Mapping into a more discriminative embedding space should lead clustering task to be easier. On the other hand, the retrieval task is directly related to the metric learning problem. It aims to retrieve similar samples from k-nearest neighborhood of the queried sample.

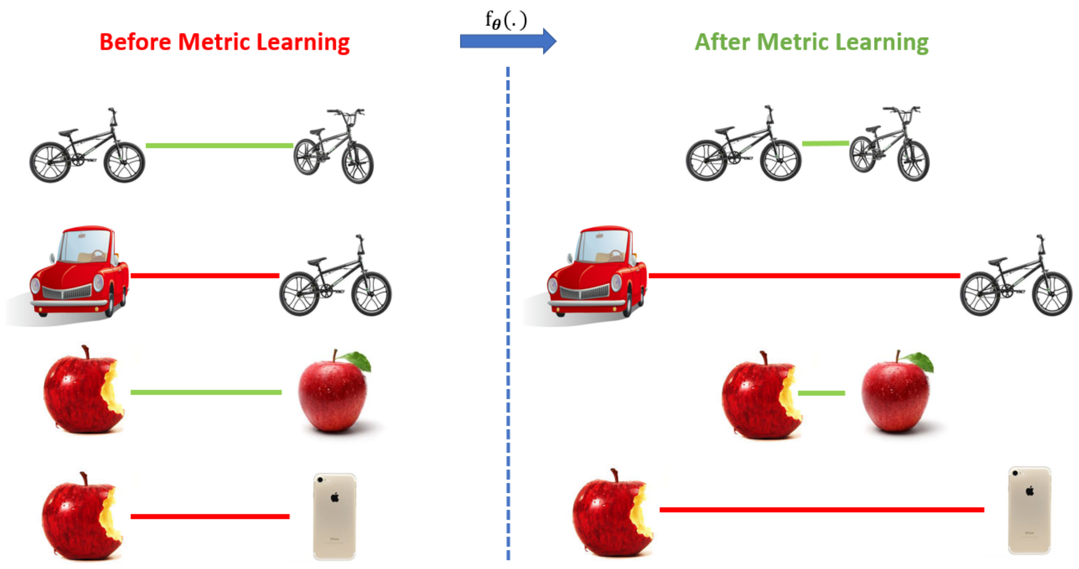


Figure 1.1: An example illustration in order to show the motivation behind metric learning approaches which learn mapping, $f_{\boldsymbol{\theta}}(.)$, such that similar examples are mapped to closer points on a embedding space and dissimilar examples are mapped to relatively far away points. Green and red lines indicate similarity and dissimilarity, respectively. (Best viewed in color.)

1.2 Motivation and Contributions

Deep metric learning methods with supervised data are studied within the scope of this thesis. The history of the deep metric learning literature is started with the contrastive loss [28] which minimizes the distance between a pair of similar examples while keeping a pair of dissimilar samples away from each other. Later, triplet loss [23, 29] is proposed that a set of triplets are established instead of paying attention only to a pair of examples. These triplets consist of an *anchor*, a positive example of the anchor and a dissimilar sample to the anchor. Therefore, selecting representative negative examples of the triplet becomes an issue. Sohn [30] introduced N-pair loss function and sampling method to address this problem. In [30], it is suggested to construct mini-batches from N pairs of examples which belong to N different classes and minimize the loss function by utilizing (N+1)-tuplets which contain an anchor, one similar example and N-1 negative sample. However, these methods ignore the representative power of multiple positive examples to ensure including an adequate number of negatives in the mini-batch.

In order to utilize the full mini-batch information, lifted structured and embedding via facility location objective functions are also presented [1, 15]. However, they neglect the significance of selecting representative hard negatives.

Furthermore, Wang et al. [31] examined the existing deep metric learning methods in a different aspect. The authors [31] claim that the distance based loss functions consider only second-order relations in the embedding space. Hence, the angular loss is proposed in order to capture third-order correlations by utilizing triangular constraints in triplets. However, the solution space may still be higher-order.

In the scope of the above analysis, there are three main contributions in this thesis to remove these deficiencies of the existing metric learning algorithms:

1. A novel metric learning loss function, which is an extension of N-pair embedding loss, is proposed to improve the representation power of the network, since the negative examples are kept away by considering multiple similar samples instead of considering only one positive example in N-pair loss function. (Section 3.3)
2. In order to select hard negatives and eliminate outliers, a novel local neighborhood sampling method is introduced. (Section 3.4)
3. Owing to these above-mentioned loss function and sampling method, deep metric learning framework with RNN is presented to model higher-order relations in the data. (Section 4.2)

1.3 Outline of the Thesis

This thesis consists of five chapters. Following the introduction to problem definition of the metric learning task, motivation and contributions are summarized in Chapter 1.

Chapter 2 is reserved for the review of the literature on metric learning problem. At first, metric learning methods are classified into two groups and significant points of each group are detailed. The most attention is paid to deep metric learning methods owing to outstanding performances in clustering and retrieval tasks. Following that, in order to ensure comprehension, the last section is devoted to the deep neural networks.

Chapter 3 presents the proposed approach that consists of a novel deep metric learning loss function and a sampling method to address issues in Section 1.2. Following the interpretation of the motivation behind the novel deep metric learning approach, the general framework of deep metric learning based on CNN is represented. Then, each part of the proposed method is explained in detail under three main sections. Finally, the efficiency of the proposed approach is demonstrated with quantitative and qualitative experimental results.

Chapter 4 introduces a novel deep metric learning framework that contains both of CNN and RNN by utilizing loss function and sampling method which are introduced in Chapter 3. Following the explanation of the motivation behind this chapter, the novel deep metric learning framework is thoroughly described. Finally, the efficacy of the proposed framework is validated with quantitative and qualitative experimental results.

Finally, Chapter 5 highlights the important points of the proposed method for effective metric learning and draw conclusions from the experiments. Additionally, future research directions are mentioned in this chapter.

CHAPTER 2

RELATED WORK

In the scope of this thesis, the content of the background materials could be divided into two groups. First of all, the metric learning background is a good start to clearly comprehend the problem definition. Hence, conventional and novel metric learning methods are described in Section 2.1. Furthermore, due to their remarkable achievement in metric learning area [1, 9, 15, 28, 29, 30, 31, 32], the last section of the related work is reserved for deep neural networks. In Section 2.2, convolutional neural networks (CNN) and recurrent neural networks (RNN) are introduced for completeness.

2.1 Metric Learning

Learning similarities between pairs of examples has a major significant role for a range of tasks, such as clustering [9], person re-identification [10, 11, 12], face identification [13, 14], feature-based retrieval [1, 15] and challenging classification problems [16, 17, 18]. The main aim of metric learning is to learn a feature mapping such that embedding representations of similar examples are mapped to closer points on a manifold and embedding representations of dissimilar examples are mapped to relatively far away points.

Supervised metric learning approaches could be divided into two categories: linear and non-linear methods [21]. Although the goal of linear approaches is to learn linear metric such as, Mahalanobis distance [13, 22, 23], non-linear methods utilize neural networks and kernels to capture non-linear relations in the data [1, 9, 15, 28, 29, 30, 31, 32].

2.1.1 Linear Metric Learning Methods

Although linear metrics are insufficient to model non-linear variations in the data, their objective functions usually tend to be convex. Hence, their optimization is relatively simple and also robust to overfitting [21]. The most popular linear metric is the *Mahalanobis distance* which is usually exploited due to its simplicity and comprehensibility in terms of a linear projection [13, 22, 23]. The Mahalanobis distance can be described as a distance measure between two random vectors, $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$, which are assumed to be from the same distribution whose covariance matrix is equal to $\boldsymbol{S}$:

$$d_{Mah}(\boldsymbol{x}_1, \boldsymbol{x}_2) = \sqrt{(\boldsymbol{x}_1 - \boldsymbol{x}_2)^T \boldsymbol{S}^{-1} (\boldsymbol{x}_1 - \boldsymbol{x}_2)} \tag{2.1}$$

However, the term of Mahalanobis distance in metric learning literature is denoted as a generalized quadratic distance:

$$d_M(\boldsymbol{x}_1, \boldsymbol{x}_2) = \sqrt{(\boldsymbol{x}_1 - \boldsymbol{x}_2)^T \boldsymbol{M}^{-1} (\boldsymbol{x}_1 - \boldsymbol{x}_2)} \tag{2.2}$$

where $\boldsymbol{M}$ denotes trainable parameter such that $\boldsymbol{M} \in \mathbb{S}_+^d$, where $\mathbb{S}_+^d$ indicates the cone of positive semi-definite $d \times d$ square matrix whose elements are real-valued (Figure 2.1). $\boldsymbol{M}$ requires to meet this condition in order for d_M to satisfy the properties of a pseudo-distance where $\forall \boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3 \in X$:

1. $d_M(\boldsymbol{x}_1, \boldsymbol{x}_2) \geq 0$ (non-negativity)

2. $d_M(\boldsymbol{x}_1, \boldsymbol{x}_1) = 0$ (identity)

3. $d_M(\boldsymbol{x}_1, \boldsymbol{x}_2) = d_M(\boldsymbol{x}_2, \boldsymbol{x}_1)$ (symmetry)

4. $d_M(\boldsymbol{x}_1, \boldsymbol{x}_3) \leq (\boldsymbol{x}_1, \boldsymbol{x}_2) + (\boldsymbol{x}_2, \boldsymbol{x}_3)$ (triangle inequality)

It is worth mentioning that the Euclidean distance could be obtained when $\boldsymbol{M}$ is an identity matrix. If it is not, $\boldsymbol{M}$ could be defined as $\boldsymbol{L}^T \boldsymbol{L}$, where $\boldsymbol{L} \in \mathbb{R}^{kxd}$ and k denotes the rank of $\boldsymbol{M}$. Hence, $d_M(\boldsymbol{x}_1, \boldsymbol{x}_2)$ could be reformulated as:

$$d_M(\boldsymbol{x}_1, \boldsymbol{x}_2) = \sqrt{(\boldsymbol{L}\boldsymbol{x}_1 - \boldsymbol{L}\boldsymbol{x}_2)^T (\boldsymbol{L}\boldsymbol{x}_1 - \boldsymbol{L}\boldsymbol{x}_2)} \tag{2.3}$$

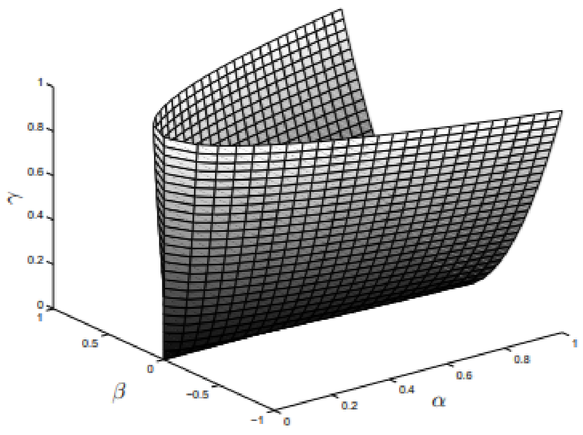


Figure 2.1: The cone $\mathbb{S}_+^2$ of positive semi-definite is described as $\begin{bmatrix} \alpha & \beta \\ \beta & \gamma \end{bmatrix}$ [21].

The Mahalanobis distance actually calculates the Euclidean distance between data points which are linearly mapped by the transformation matrix, $\boldsymbol{L}$. When $\boldsymbol{M}$ is low-rank which means $rank(\boldsymbol{M}) = k \leq d$, the dimension of embedding vectors is reduced from d to k. Therefore, the data is linearly mapped from a higher dimensional space to a lower dimensional space. It contributes to decrease in computational complexity. Due to these great properties of the Mahalanobis distance, it is one of the most preferred linear metric distances in the literature.

In spite of these nice properties of Mahalanobis distance, there are two noticeable challenges during its learning phase. First of all, it is hard to ensure $\boldsymbol{M} \in \mathbb{S}_+^d$ constraint. The projected gradient method, which minimizes an objective function subject to a constraint, is usually utilized to satisfy constraint that by setting the negative eigenvalues to zero [36]. Nevertheless, since the complexity of eigenvalue decomposition is $O(d^3)$, the computational cost of the high-dimensional data is expensive. Then, learning a low-rank matrix remains as the second challenge in order to provide dimensionality reduction. The reason of that challenge is due to the fact that optimizing $\boldsymbol{M}$, while maintaining a rank constraint, is an NP-hard problem [21].

One of the most popular methods to optimize $\boldsymbol{M}$ is the Large Margin Nearest Neighbors (LMNN) algorithm proposed by Weinberger et al. [23]. In this algorithm, the optimization problem is solved by adding local constraints which are defined through considering k nearest neighbors. The method attempts to keep similar classes in k

nearest neighborhood while pushing dissimilar classes away. The distance metric could be learned as below:

$$\begin{aligned}
&\{\min_{\boldsymbol{M}\in\mathbb{S}_+^d}(1-\mu)\sum_{(\boldsymbol{x_i},\boldsymbol{x_j})\in\mathbb{P}} d_M^2(\boldsymbol{x_i},\boldsymbol{x_k})+\mu\sum_{i,j,k}\xi_{i,j,k}\}\\
s.t.\;\; &d_M^2(\boldsymbol{x_i},\boldsymbol{x_k})-d_M^2(\boldsymbol{x_i},\boldsymbol{x_j})\geq 1-\xi_{i,j,k} \qquad \forall(\boldsymbol{x_i},\boldsymbol{x_j},\boldsymbol{x_k}\in\mathbb{N})\\
&\mathbb{P}=\{(\boldsymbol{x_i},\boldsymbol{x_j}):y_i=y_j \text{ and } \boldsymbol{x_j}\in \text{ k-neighborhood of } \boldsymbol{x_i}\}\\
&\mathbb{N}=\{(\boldsymbol{x_i},\boldsymbol{x_j},\boldsymbol{x_k}):(\boldsymbol{x_i},\boldsymbol{x_j})\in\mathbb{P}, y_i\neq y_k\}
\end{aligned} \tag{2.4}$$

where $\mu \in [0,1]$ is control variable that provides a "pull/push" trade-off. $\xi_{i,j,k}$ denotes a slack variable which is a parameter that is used to convert the inequality constraint to the equality constraint. This objective function is solved by *sub-gradient descent* [37] and *book-keeping* [38] which handles billions of constraints. Although LMNN suffers from overfitting due to a lack of regularization term, LMNN works well in practice.

Davis et al. [22] proposed Information-Theoretic Metric Learning (ITML) method that presents LogDet divergence regularization in order to prevent overfitting. This Bregman divergence is described in the following equation:

$$D_{log-det}(\boldsymbol{M},\boldsymbol{M}_0)=\text{tr}(\boldsymbol{M}\boldsymbol{M}_0^{-1})-\log\det(\boldsymbol{M}\boldsymbol{M}_0^{-1})-d \tag{2.5}$$

where d indicates the dimension of the input space. $\boldsymbol{M}_0$ is some positive definite matrix which is desired to be close to $\boldsymbol{M}$. $\boldsymbol{M}_0$ is usually selected as an identity matrix, $\boldsymbol{I}$, in order not to diverge from the Euclidean distance. The most significant property of the LogDet divergence is that it is finite only when $\boldsymbol{M}$ is positive definite. Hence, minimizing $D_{log-det}$ guarantees that $\boldsymbol{M}$ keeps positive semi-definiteness without adding any constraint. ITML method could be defined as:

$$\begin{gathered}
\{\min_{\boldsymbol{M}\in\mathbb{S}^d_+} \boldsymbol{D}_{log-det}(\boldsymbol{M},\boldsymbol{M}_0)+\gamma\sum_{i,j}\xi_{i,j}\} \\
s.t.\ \ d^2_M(\boldsymbol{x_i},\boldsymbol{x_j}) \le u+\xi_{i,j} \qquad \forall(\boldsymbol{x_i},\boldsymbol{x_j}\in\mathbb{P}) \\
d^2_M(\boldsymbol{x_i},\boldsymbol{x_j}) \ge v-\xi_{i,j} \qquad \forall(\boldsymbol{x_i},\boldsymbol{x_j}\in\mathbb{N}) \\
\mathbb{P}=\{(\boldsymbol{x_i},\boldsymbol{x_j}):y_i=y_j\} \\
\mathbb{N}=\{(\boldsymbol{x_i},\boldsymbol{x_j}):y_i\neq y_j\}
\end{gathered} \tag{2.6}$$

where u, v denote threshold values. γ is a trade-off parameter that is greater than zero. The main goal of ITML is accomplishing similarity and dissimilarity constraints by preserving $\boldsymbol{M}$ as close as possible to the Euclidean distance which means that $\boldsymbol{M}_0$ is an identity matrix. Furthermore, minimizing $\boldsymbol{D}_{log-det}$ equals to minimizing the *KL divergence* [39] between two multivariate Gaussian distributions which are constructed by $\boldsymbol{M}$ and $\boldsymbol{M}_0$. The minimization problem which is defined in (2.6) could be solved as in [22]. Although ITML yields good results in the literature, the performance highly depends on the selected value of $\boldsymbol{M}_0$.

2.1.2 Non-linear Metric Learning Methods

The drawback of the conventional linear metric learning approaches, which are described in Section 2.1.1, is that they are unable to learn non-linear relations in the data. Kernel tricks are usually utilized to address this issue [40, 41, 42, 43]. However, since selecting a kernel is quite empirical, the generalization power of these methods is limited. After observing the remarkable performance of deep learning in various machine learning tasks [24, 25, 26, 27], deep metric learning methods are also proposed [1, 9, 15, 28, 29, 30, 31, 32]. These recent works on deep metric learning is summarized in this section.

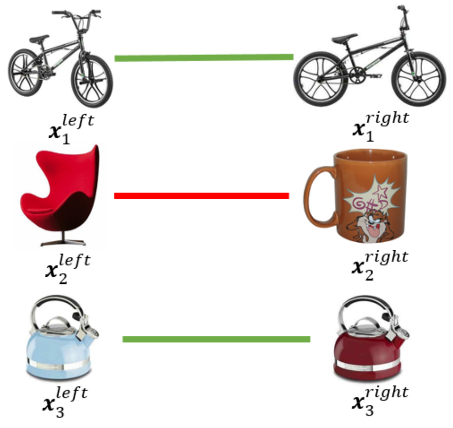
2.1.2.1 Contrastive Embedding

The fundamental idea behind contrastive embedding is minimizing the distance between a positive pair of examples and maximizing the distance between a pair of examples with the different class label. The objective function is described as follows [28]:

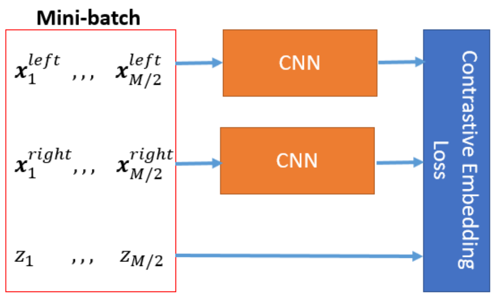
$$J_{cont-emb} = \frac{1}{M} \sum_{(i,j)}^{M/2} z_{i,j} D_{i,j}^2 + (1 - z_{i,j})[\alpha - D_{i,j}]_+^2 \tag{2.7}$$

$$D_{i,j} = \|f(\boldsymbol{x}_i) - f(\boldsymbol{x}_j)\|_2 \tag{2.8}$$

where M denotes batch size, $D_{i,j}$ is distance between paired data, $\boldsymbol{x_i}$ and $\boldsymbol{x_j}$, and $f(.)$ indicates the non-linear function which is learned by the neural network. If the class label, $z_{i,j}$, is 1, $\boldsymbol{x_i}$ and $\boldsymbol{x_j}$ are from the same class. If the class of $\boldsymbol{x_i}$ is not same with $\boldsymbol{x_j}$, vice versa. The hinge function, $max(0, \cdot)$, is represented by $[\cdot]$ operator. Finally, the margin, α, stands in order to penalize distances between a pair of negatives which are smaller than α. The training process is illustrated in Figure 2.2.



(a)


Mini-batch
x_1^{left} ,,, $x_{M/2}^{left}$
x_1^{right} ,,, $x_{M/2}^{right}$
z_1 ,,, $z_{M/2}$
CNN
CNN
Contrastive Embedding Loss

(b)

Figure 2.2: (a) A typical visualization of mini-batch construction with six examples for contrastive learning. Green and red lines indicate similarity and dissimilarity, respectively. (b) Training CNN architecture with contrastive embedding loss. (Best viewed in color.)

2.1.2.2 Triplet Embedding

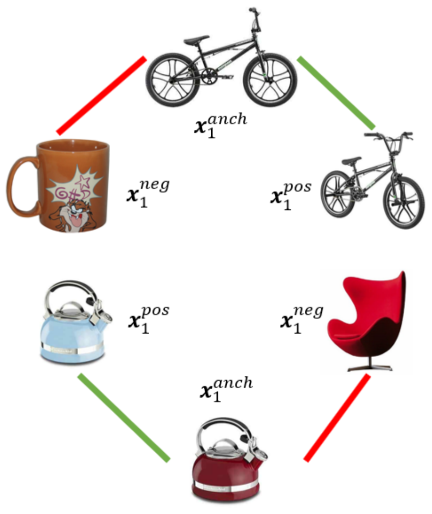
Triplet embedding loss is an advancement on contrastive embedding loss [23, 29]. Instead of paying attention to a pair of examples, a set of triplets are established.

Each triplet consists of an anchor, a positive and a negative example. While an anchor has the same class label with a positive example, a negative one has a distinct class label. A sample of triplet construction and training in mini-batch with six examples is illustrated in Figure 2.3. The aim of the method is that the embedding vector of positive example is getting closer to the embedding representation of an anchor and the distance between embeddings of an anchor and a negative example is getting larger. The triplet embedding loss function could be defined in following equation:

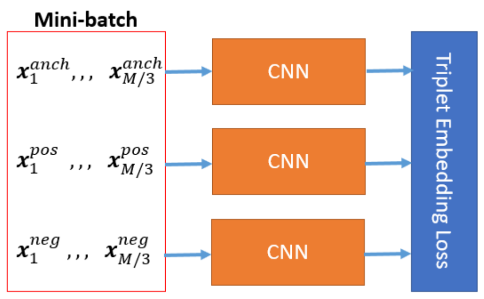
$$J_{triplet} = \frac{1}{|\mathbb{T}|} \sum_{(i,j,k)\in\mathbb{T}} [D^2_{i,j} - D^2_{i,k} + \alpha]_+ \tag{2.9}$$

$$D_{i,j} = \|f(\boldsymbol{x}_i) - f(\boldsymbol{x}_j)\|_2 \tag{2.10}$$

where $\mathbb{T}$ is the set of triplets. Subscripts i, j and k denotes an anchor, a positive and a negative example, respectively. $f(.)$ is the mapping trained by the neural network. The operator $[\cdot]$ stands for a hinge function which maps a negative argument to zero, while α represents a constant margin.



(a)



Mini-batch
$\boldsymbol{x}_1^{anch}$,,, $\boldsymbol{x}_{M/3}^{anch}$
$\boldsymbol{x}_1^{pos}$,,, $\boldsymbol{x}_{M/3}^{pos}$
$\boldsymbol{x}_1^{neg}$,,, $\boldsymbol{x}_{M/3}^{neg}$
CNN
CNN
CNN
Triplet Embedding Loss

(b)

Figure 2.3: (a) An illustration of mini-batch construction with six examples for triplet embedding learning. Green and red lines indicate similarity and dissimilarity, respectively. (b) Training CNN architecture with triplet embedding loss. (Best viewed in color.)

In real applications, triplet sampling strategy has a key role in obtaining good results. Schroff et al. [29] suggest that triplet could be constructed by selecting "semi-hard" negative example for each positive pair in mini-batch. The embedding distance between "semi-hard" negative example, k, and an anchor, i, should be larger than the distance of a pair constructed by an anchor, i, and a positive example, j. However, it is still hard, since this distance is the closest one to the $D_{i,j}^2$. Hence, the cost function could be reformulated as:

$$J_{triplet} = \frac{1}{|\mathbb{P}|} \sum_{(i,j)\in\mathbb{P}} [D_{i,j}^2 - D_{i,\hat{k}(i,j)}^2 + \alpha]_+ \tag{2.11}$$

$$\hat{k}(i,j) = \underset{k\in \boldsymbol{z}[k]\neq \boldsymbol{z}[i]}{\arg\min}\ D_{i,k}^2 \quad \text{s.t.} \quad D_{i,k}^2 \geq D_{i,j}^2 \tag{2.12}$$

where $\mathbb{P}$ is the set of positive pairs. If $\hat{k}(i,j)$ could not be found in Equation 2.12, the furthest negative example is selected as below:

$$\hat{k}(i,j) = \underset{k\in \boldsymbol{z}[k]\neq \boldsymbol{z}[i]}{\arg\max}\ D_{i,k}^2 \tag{2.13}$$

FaceNet method [29] utilizes huge mini-batches which contain 1800 images in order to get good performance. Quite large mini-batches are required to pick correct semi-hard negatives. Due to the memory capacity of GPU, the method makes it difficult to train some networks.

2.1.2.3 Lifted Structured Embedding

Two methods which are described in Sections 2.1.2.1 and 2.1.2.2 build training mini-batches by creating pairs and triplets, respectively. Since these methods do not exploit all information from a batch, Song et al. [15] presented lifted structured embedding loss. This method utilizes entire pairwise distances within a batch as illustrated in Figure 2.4(a). The objective function of lifted structured embedding that consists of all positive and negative pairs within a batch could be described as follows:

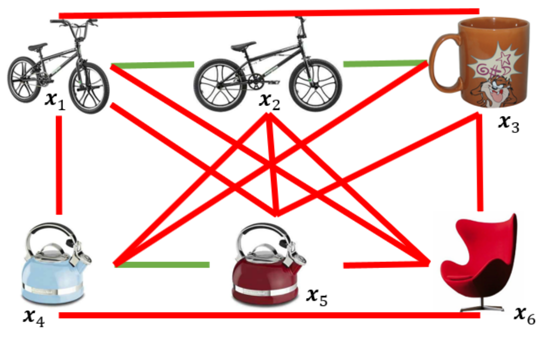
$$J_{lifted} = \frac{1}{2|\mathbb{P}|} \sum_{(i,j)\in\mathbb{P}} [J_{i,j}]_+^2 \tag{2.14}$$

$$J_{i,j} = D_{i,j} + \max\left(\max_{(i,l)\in\mathbb{N}} (\alpha - D_{i,l}), \max_{(j,k)\in\mathbb{N}} (\alpha - D_{j,k}) \right) \tag{2.15}$$

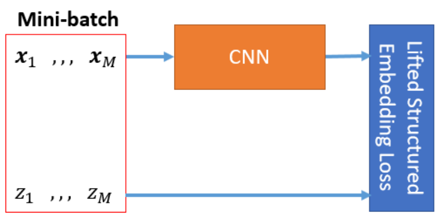
$$D_{i,j} = \|f(\boldsymbol{x}_i) - f(\boldsymbol{x}_j)\|_2 \tag{2.16}$$

where $\mathbb{P}$ and $\mathbb{N}$ indicate sets of positive and negative pairs in training batch. $f(.)$ is the function that represents a trainable network. The desired margin value is shown as α. Although the aim of the objective function defined in Equation 2.14 is obvious, it is non-smooth and mining all negative and positive pairs in nested max functions is too expensive. In order to obtain a differentiable smooth objective functions, the log-sum-exp formulation is utilized while selecting hard negative pairs:

$$\hat{J}_{lifted} = \frac{1}{2|\mathbb{P}|} \sum_{(i,j)\in\mathbb{P}} [D_{i,j} + \log(\sum_{(i,l)\in\mathbb{N}} \exp\{\alpha - D_{i,l}\} + \sum_{(j,k)\in\mathbb{N}} \exp\{\alpha - D_{j,k}\})]_+^2 \tag{2.17}$$



(a)



(b)

Figure 2.4: (a) An visualization of mini-batch construction with six examples for lifted structured embedding learning. Green and red lines indicate similarity and dissimilarity, respectively. (b) Training CNN architecture with lifted structured embedding loss. (Best viewed in color.)

It should be noted that multiple CNN networks are no longer required as seen in Figure 2.4(b) due to the log-sum-exp formulation which is defined in Equation 2.17.

2.1.2.4 Multi-class N-Pair Embedding

Although the aforementioned methods which are described in the previous subsections yield good results, they still suffer from undesired local minima and slow convergence due to their hard negative mining methods. Elegant hard negative mining search algorithms may address this issue but they are quite expensive to apply. In [30], the authors propose an $(N+1)$-tuplet loss which emphasizes one similar example among $N-1$ dissimilar examples. It is an extension to triplet loss described in Section 2.1.2.2. When $N=2$, N-pair embedding loss is equal to triplet embedding loss. The $(N+1)$-tuplet loss function is proposed as [30]:

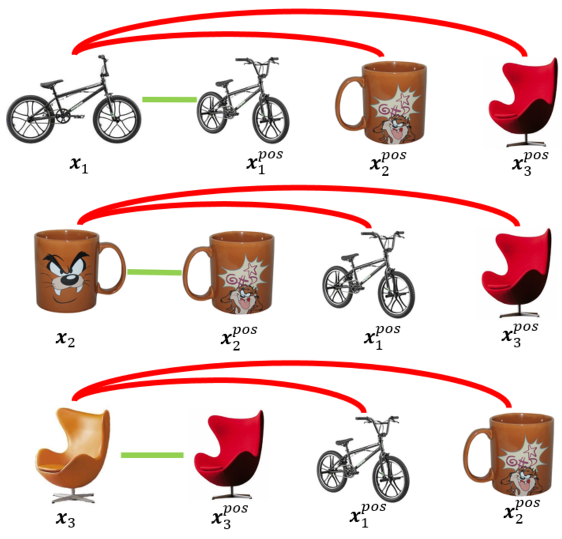
$$J_{n-pair} = \log(1 + \sum_{(i,k)\in\mathbb{N}} \exp\{D_{i,k} - D_{i,j}\}) \tag{2.18}$$

$$D_{i,j} = f(\boldsymbol{x}_i)^T f(\boldsymbol{x}_j) \tag{2.19}$$

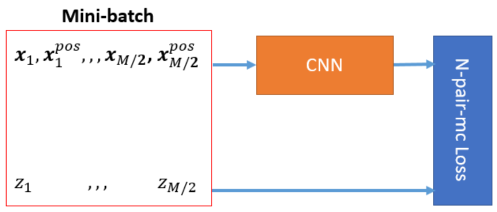
where $(i,j) \in \mathbb{P}$ and $f(.)$ is the embedding function which is learned by the neural network. $\mathbb{P}$ and $\mathbb{N}$ represent sets of positive and negative pairs in mini-batch. The desired loss function described in Equation 2.18 is quite complex when the number of output classes, L, is large. Even if the number of dissimilar samples per class is restricted to one, it is still impossible to train, since the size of mini-batch should be equal to at least L. Nevertheless, ideal $(L+1)$-tuplet loss is worth to define:

$$\log(1 + \sum_{(i,k)\in\mathbb{N}} \exp\{D_{i,k} - D_{i,j}\}) = -\log \frac{\exp\{D_{i,j}\}}{\exp\{D_{i,j}\} + \sum_{k=1}^{L-1} \exp\{D_{i,k}\}} \tag{2.20}$$

The loss function which is defined above resembles softmax loss that is multi-class logistic loss function. It suggests that N-pair loss is a more generalized version of triplet loss.



(a)



(b)

Figure 2.5: (a) An illustration of mini-batch construction with eight examples for N-pair embedding learning. Green and red lines indicate similarity and dissimilarity, respectively. (b) Training CNN architecture with multi-class N-pair embedding loss. (Best viewed in color.)

In order to get rid of the computational burden of the N-pair embedding objective function, one positive pair of similar examples is randomly picked while constructing a mini-batch. After that, the negative examples which contain pairs of similar samples that violate the triplet condition indicated in Equation 2.12 and 2.13 are selected at random as hard negatives. The illustration of mini-batch construction is indicated in Figure 2.5. Hence, the multi-class N-pair loss function could be reformulated as:

$$J_{n-pair-mc} = \frac{-1}{|\mathbb{P}|} \sum_{(i,j)\in\mathbb{P}} \log \frac{\exp\{D_{i,j}\}}{\exp\{D_{i,j}\} + \sum_{(i,k)\in\mathbb{N}} \exp\{D_{i,k}\}} \tag{2.21}$$

Moreover, the objective function described in Equation 2.21 is minimized by not only the direction of $D_{i,j}$ but also its magnitude. However, only the direction is significant to determine the distance between examples. Although normalization seems like a solution, forcing $D_{i,j}$ to be less than 1 makes optimization difficult. Instead of that,

L_2 regularization term is added on loss:

$$J_{n-pair-mc-reg} = J_{n-pair-mc} + \frac{\lambda}{M}\sum_{i=1}^{M}\|f(\boldsymbol{x_i})\|_2 \tag{2.22}$$

where M and λ denote mini-batch size and regularization constant, respectively.

2.1.2.5 Angular Embedding

Wang et al. [31] put forth the disadvantages of relying on a certain distance metric between similar and dissimilar examples in a training loss function. First of all, proposed distance metrics have not robustness to scale change. The mentioned metric learning algorithms enforce a distance gap between examples which belong to distinct classes. Selecting the same constant margin for all clusters overlooks different scales of intra-class variation. Secondly, minimizing loss function by only taking distances between examples into account, considers second-order information of the dataset and neglects high-order correlations.

In order to overcome these issues, the third-order relation is added to conventional triplet loss by restricting the upper bound of the angle at the negative example of the triplet. This constraint provides keeping negative examples far away from anchors and dragging positive examples closer to each other. This similar idea is utilized to enhance pairwise constraints in Markov random fields [44]. The angular loss is defined as belows:

$$J_{angular} = \frac{1}{|\mathbb{T}|}\sum_{(i,j,k)\in\mathbb{T}}[D_{i,j}^2 - 4\tan^2\alpha D_{k,c}^2]_+ \tag{2.23}$$

$$D_{i,j} = \|f(\boldsymbol{x}_i) - f(\boldsymbol{x}_j)\|_2 \tag{2.24}$$

where $\mathbb{T}$ denote the set of triplets. Subscripts i, j and k represent an anchor, a positive and a negative example, respectively. Center point, $\boldsymbol{x_c}$, is calculated as $\boldsymbol{x_c} = (\boldsymbol{x_i}+\boldsymbol{x_j})/2$. $f(.)$ is the function trained by the neural network. α stands for predefined upper bound as before. Since N-pair loss and angular loss consider second and third order information of dataset, respectively, the joint optimization with the angular and N-pair loss that is described in Equation 2.22 increases the performance of the network:

$$J_{angular\&n-pair} = J_{n-pair-mc-reg} + \gamma J_{angular} \tag{2.25}$$

where γ is a trade-off parameter between these two objective functions.

2.1.2.6 Embedding via Facility Location

The common property of the metric learning algorithms which are mentioned in the above subsections can be summarized as being local methods. Figure 2.6 indicates the weak side of the local metric learning methods. Whenever a pair of similar examples is separated by dissimilar examples, repulsive gradient signals which are constructed by negative pairs may be larger than an attractive gradient signal created by a positive pair. This failure could cause many similar examples to be mapped in far away points of the embedding space. In order to remove this problem, facility location embedding loss that includes a clustering loss is proposed [15].

Let $\boldsymbol{X} = [\boldsymbol{x}_1, ..., \boldsymbol{x}_N]$ be a set of inputs and $f(\boldsymbol{x}_i; \theta)$ be a non-linear function which maps each input into an embedding space. The aim of facility location is mapping each input to its closest point from a selected set of landmarks, $\mathbb{S} \subseteq \mathbb{G}$, where $\mathbb{G} = \{1, ..., |\boldsymbol{X}|\}$ denotes the ground-truth set. The facility location function is defined by the following equation [15]:

$$F(\boldsymbol{X}, \mathbb{S}; \theta) = -\sum_{i \in |\boldsymbol{X}|} \min_{j \in \mathbb{S}} \| f(\boldsymbol{x}_i; \theta) - f(\boldsymbol{x}_j; \theta) \| \tag{2.26}$$

The main idea behind facility location function is determining the sum of the distances between each customer in $\boldsymbol{X}$ and their closest facility location in $\mathbb{S}$. In the clustering problem, $\mathbb{S}$ represents cluster medoids and clustering is performed according to the closest medoid from each example point. Although optimizing Equation 2.26 is NP-hard, a greedy approach with sub-modularity exits in the literature [3].

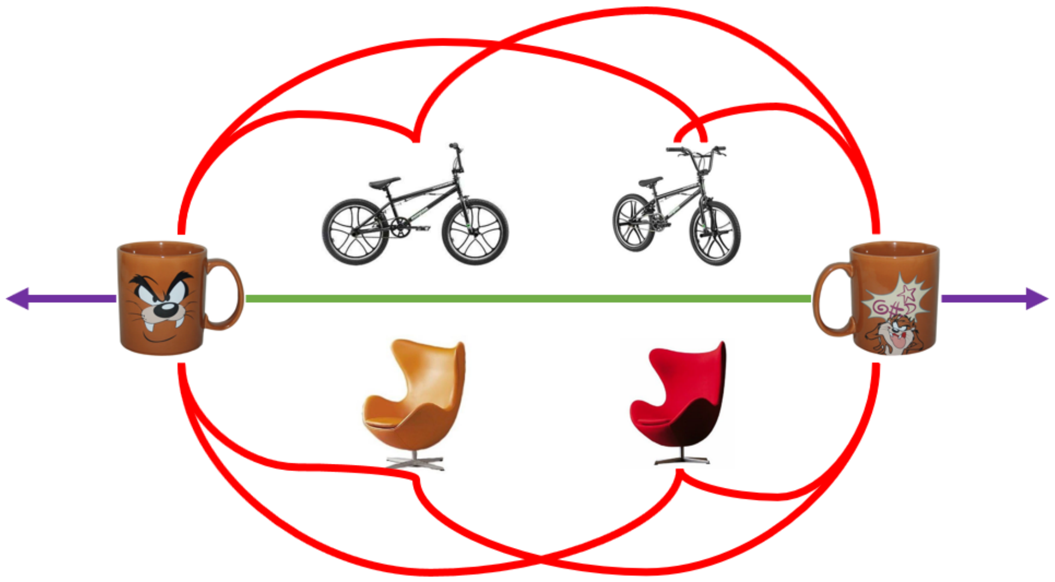


Figure 2.6: A failure example for local metric learning methods. Since repulsive gradient signals generated by negative pairs illustrated in red lines is larger than an attractive gradient signal created by a positive pair denoted as a green line, two similar examples cannot be clustered correctly. (Best viewed in color.)

An oracle scoring function, $\hat{F}$, evaluates the quality of the clustering assignment with respect to given ground-truth cluster labels, $\boldsymbol{z}^*$:

$$\hat{F}(\boldsymbol{X}, \boldsymbol{z}^*; \theta) = \sum_{k}^{|\boldsymbol{Z}|} \max_{j \in \{i : \boldsymbol{z}^*[i] = k\}} F(\boldsymbol{X}_{\{i : \boldsymbol{z}^*[i] = k\}}, \{j\}; \theta) \tag{2.27}$$

The oracle scoring function should be larger than the clustering score which is obtained by the worst clustering assignment. Therefore, facility location embedding function can be defined as:

$$J_{facility} = [\max_{\mathbb{S} \in \mathbb{G},\ |\mathbb{S}| = |\boldsymbol{Z}|} (F(\boldsymbol{X}, \mathbb{S}; \theta) + \gamma C(g(\mathbb{S}, \boldsymbol{z}^*))) - \hat{F}(\boldsymbol{X}, \boldsymbol{z}^*; \theta)]_+ \tag{2.28}$$

where $\boldsymbol{z} = g(\mathbb{S})$ denotes the function that maps to set of cluster labels by assigning each sample point to the closest facility in $\mathbb{S}$:

$$g(\mathbb{S})[i] = \arg\min_j \|f(\boldsymbol{x}_i;\theta) - f(\boldsymbol{x}_{\{j:j\in\mathbb{S}\}};\theta)\| \tag{2.29}$$

The structured margin term, $C(g(\boldsymbol{z},\boldsymbol{z}^*))$, is evaluation metric for clustering. When clustering is performed perfectly, the output of this metric is 0. Besides, the poorest clustering quality is obtained if $C(g(\boldsymbol{z},\boldsymbol{z}^*))$ is 1. The utilized margin term is described below:

$$C(g(\boldsymbol{z},\boldsymbol{z}^*)) = 1 - NMI(\boldsymbol{z},\boldsymbol{z}^*) \tag{2.30}$$

where NMI stands for the normalized mutual information (NMI) [45]. NMI determines the clustering quality between two clustering assignments:

$$NMI(\boldsymbol{z},\boldsymbol{z}^*) = \frac{I(\boldsymbol{z};\boldsymbol{z}^*)}{\sqrt{H(\boldsymbol{z})H(\boldsymbol{z}^*)}} \tag{2.31}$$

where $H(.)$ and $I(.,.)$ denote entropy and mutual information functions, respectively. $H(.)$ and $I(.,.)$ can be calculated by utilizing the marginal, $P(i)$, and joint probability, $P(i,j)$, mass functions:

$$P(i) = \frac{1}{N}\sum_j \mathbb{I}[\boldsymbol{z}[j] == i] \tag{2.32}$$

$$P(i,j) = \frac{1}{N}\sum_{k,l} \mathbb{I}[\boldsymbol{z}[k] == i]\mathbb{I}[\boldsymbol{z}[l] == j] \tag{2.33}$$

where N is the number of samples. $\mathbb{I}[.]$ represents indicator function whose output is one if the condition, [.], satisfies. If not, $\mathbb{I}[.]$ outputs zero.

To sum up, since facility location embedding loss realizes the global landscape of the embedding space, it can escape the poor local optima which is illustrated in Figure 2.6 via pulling all positive examples to cluster medoids and pushing negative examples further from medoids by the structured margin term.

2.2 Deep Neural Networks

Artificial neural networks (ANN) can be said to mimic the activity of human brains that consist of biological neural networks [5]. These systems *learn* how to solve specific problems instead of applying rule-based methods. Although the history of artificial neural networks dates back to 1940s [46], it became very popular after the computation power of computers is enhanced and the number of available training data is increased. These innovations lead to building networks which have more layers. For this reason, the term of the artificial neural network is replaced with deep neural network.

Deep neural networks have acquired a lot of attention with its accomplishment in a variety of tasks such as object detection [47, 24, 48], object instance segmentation [49, 50], image captioning [51, 52, 53], face recognition [29, 54], natural language processing [55, 56, 57], speech recognition [58, 59], etc. In this scope of thesis, convolutional and recurrent neural network models are focused. Nevertheless, due to integrity of the comprehension, the basic and advanced concepts of deep neural networks related to this thesis are explained in below subsections.

2.2.1 Deep Feedforward Networks

Deep feedforward networks which is also known as multilayer perceptrons (MLP), is typical deep neural network models. Multilayer perceptrons are utilized to estimate some function, f. It could be described as a function, $\boldsymbol{y} = f(\boldsymbol{x}; \boldsymbol{\theta})$, that maps an input, $\boldsymbol{x}$, to an output, $\boldsymbol{y}$, by learning network parameters, $\boldsymbol{\theta}$, which provides the best approximation to the desired function. As the name of the model implies, there are no feedback connections between the previous output and current computation.

More than one functions constitute deep feedforward networks which are a directed acyclic graph. For instance, if MLP has two *hidden layers* that is illustrated in Figure 2.7, the output function of the network could be described as:

$$f(\boldsymbol{x}) = f^{(o)}(f^{(2)}(f^{(1)}(\boldsymbol{x}))) \tag{2.34}$$

In Equation 2.34, $f^{(1)}$, $f^{(2)}$ and $f^{(o)}$ denote first layer, second layer and output layer of the network, respectively. In the training procedure, only the output of the final layer is exploited to approximate the desired function according to some loss function. Hence, $f^{(1)}$ and $f^{(2)}$ are called as hidden layers. As seen in Figure 2.7, each hidden layer of the neural network is a vector and each element is entitled as a *hidden unit*.

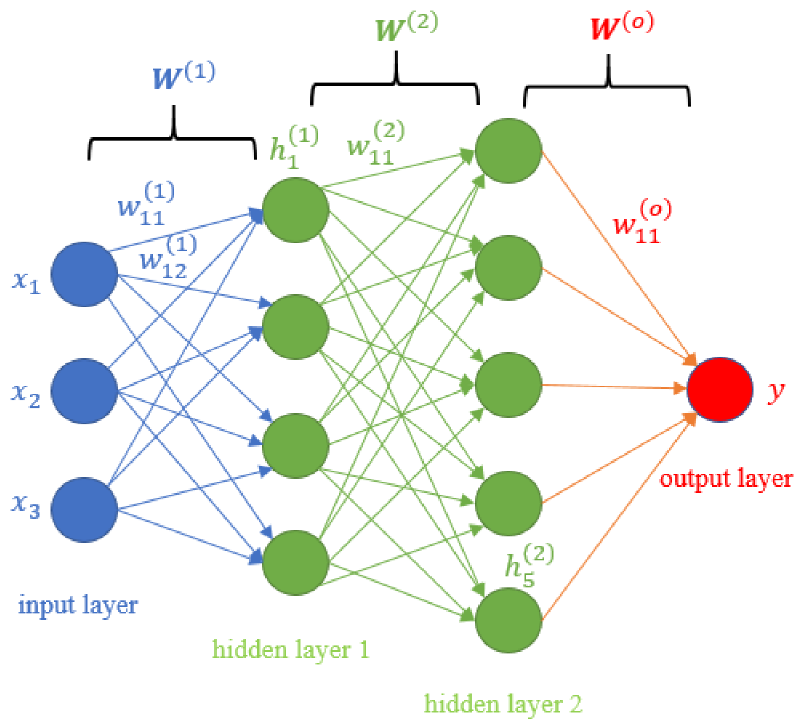


Figure 2.7: An example of multilayer perceptrons with two hidden layers. (Best viewed in color.)

Determining which functions, $f^{(i)}$, which are utilized to compute the neural network is very significant. Since linear models have limited capacity to represent the model, it is required to extend linear models by applying non-linear functions to hidden vectors, $\boldsymbol{h}^{(i)}$:

$$\boldsymbol{h}^{(i)} = \boldsymbol{W}^{(i)}\boldsymbol{x} + \boldsymbol{b}^{(i)} \tag{2.35}$$

$$f^{(i)}(\boldsymbol{x}) = \phi(\boldsymbol{h}^{(i)}) \tag{2.36}$$

where $\boldsymbol{W}^{(i)}$ and $\boldsymbol{b}^{(i)}$ describe weight matrix and bias vector, respectively. The index of layer is represented as i. The non-linear function, $\phi(.)$ is also called as an activation function.

2.2.2 Loss Functions

The aim of training neural networks is to estimate a distribution $p(\boldsymbol{y}|\boldsymbol{x};\theta)$. In most cases, the deep neural network tasks could be formulated as a classification problem. For example, the softmax function is the most popular one for a multi-class classification task. It is usually exploited in the final layer of the neural network to model a probability distribution over n different classes. In order to generalize multi-class classification problem, an output vector, $\hat{\boldsymbol{y}}$, whose elements are $\hat{y}_i = P(y = i|\boldsymbol{x})$ should be obtained. First of all, unnormalized log-probabilities which is also called as logits are acquired by applying a linear layer:

$$\boldsymbol{s} = \boldsymbol{W}^{(o)}\boldsymbol{h}^{(f)} + \boldsymbol{b}^{(o)} \tag{2.37}$$

$$s_i = \hat{P}(y = i|\boldsymbol{x}) \tag{2.38}$$

where $\boldsymbol{W}^{(o)}$ and $\boldsymbol{b}^{(o)}$ describe weight matrix and bias vector which are used to linearly map from last hidden layer, $\boldsymbol{h}^{(f)}$ into an output layer. In order to satisfy the properties of the valid probability distribution, each element of $\boldsymbol{s}$, s_i is required to be valued between 0 and 1. Besides, the sum of each s_i should be 1. Hence, the softmax function which is illustrated in Figure 2.8 is defined as:

$$\text{softmax}(\boldsymbol{s})_i = \frac{\exp z_i}{\sum_j \exp z_j} \tag{2.39}$$

During the training process, it is generally assumed that the target probability distribution, $\boldsymbol{p}$, is known. The loss function is determined to minimize the difference between $\boldsymbol{p}$ and the estimated probability distribution, $\boldsymbol{q}$. In order to compare these

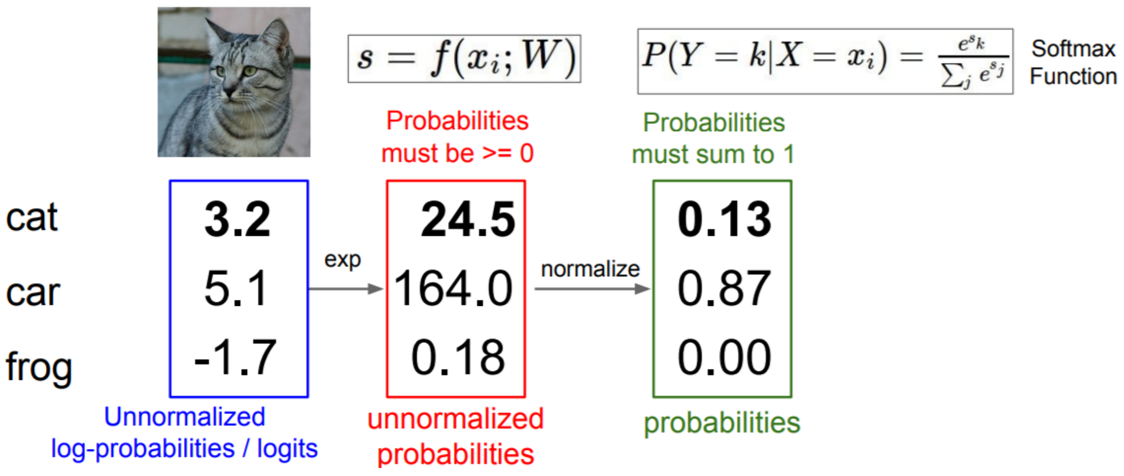

$s = f(x_i; W)$
$P(Y = k|X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}}$
Softmax Function
Probabilities must be >= 0
Probabilities must sum to 1
cat
car
frog
3.2
5.1
-1.7
exp
24.5
164.0
0.18
normalize
0.13
0.87
0.00
Unnormalized log-probabilities / logits
unnormalized probabilities
probabilities

Figure 2.8: An example of the softmax classifier for three classes case [4]. (Best viewed in color.)

distributions that are assumed to be discrete, cross entropy is utilized:

$$H(\boldsymbol{p}, \boldsymbol{q}) = -\sum_{i=1}^{C} p_i \log q_i \tag{2.40}$$

where C denotes the number of classes. The general definition of the cross-entropy could be written as in terms of the entropy, $H(.)$ and the Kullback-Leibler (KL) divergence, $D_{kl}(.||.)$,:

$$H(\boldsymbol{p}, \boldsymbol{q}) = -\mathbb{E}_p[-\log \boldsymbol{q}] = H(\boldsymbol{p}) + D_{kl}(\boldsymbol{p}||\boldsymbol{q}) \tag{2.41}$$

The KL divergence is a non-symmetric measure, $D_{kl}(p||q) \neq D_{kl}(q||p)$, of the difference between two probability distributions, p and q:

$$D_{kl}(p||q) = \sum_{i=1}^{C} p_i \log \frac{p_i}{q_i} \tag{2.42}$$

As seen in Equation 2.41, since the entropy of the true distribution, $H(\boldsymbol{p})$, is a constant, minimizing the KL divergence equals to minimize the cross-entropy. Hence, the multi-class loss with softmax score function could be formulated as:

$$J(\theta) = -\sum_{i=1}^{C} y_j \log \frac{\exp z_i}{\sum_j \exp z_j} \tag{2.43}$$

where desired output, y_j, is 1 at only the index of the correct class, k. The above summation is zero at other indexes, $j \neq k$. The loss function that is defined in Equation 2.43 could be summarized as follows:

$$J(\theta) = -\log \frac{\exp z_k}{\sum_j \exp z_j} \tag{2.44}$$

Besides, the multi-class loss function with softmax could be extended for a dataset which has N samples:

$$J(\theta) = -\sum_{i=1}^{N} \log \frac{\exp z_k^i}{\sum_j \exp z_j^i} \tag{2.45}$$

where i superscript indicates the index of the sample.

2.2.3 Optimization

Once the loss function is determined, it is required to get neural network parameters that minimize it. In the most deep learning applications, gradient-based optimization is utilized. Assume that $y = f(x)$ is the function which is requested to minimize where x and y are real numbers. The derivative of this function, $f^{'}(x)$, equals to the slope of $f(x)$ at the point x. This slope indicates how to alter the input to obtain a small enhancement in y: $f(x + \epsilon) = f(x) + \epsilon f^{'}(x)$. Hence, it is guaranteed that if ϵ is small adequately, $f(x - \epsilon\, \text{sign}(f^{'}(x)))$ is always less than $f(x)$ where sign(.) is a signum function. Therefore, the loss, y, could be decreased by moving x in small steps with negative of the gradient. This optimization method is called as a gradient descent [60]. The illustration of the gradient descent technique is indicated in Figure 2.9.

Three kinds of the gradient descent technique are mainly exploited in the deep learning area. They are distinguished according to the usage of the dataset while computing the gradient of the loss function. These differences provide a trade-off between an accuracy of the model and training time.

At first place, a batch gradient descent which is as known as a vanilla gradient descent. uses whole training data to calculate the gradient of the objective function, $J(\boldsymbol{\theta})$, with respect to the neural network parameters, $\boldsymbol{\theta}$:

$$\boldsymbol{\theta} = \boldsymbol{\theta} - \eta \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \tag{2.46}$$

where η is the learning rate which determines the size of the steps to reach a local minimum. Vanilla gradient descent guarantees to reach a local minimum . If the loss function is convex, it is always converged to a global minimum. However, since vanilla gradient descent utilizes an entire dataset for one update, it is slow and inapplicable to large datasets due to the memory constraints. Besides, it is not suitable for an online training.

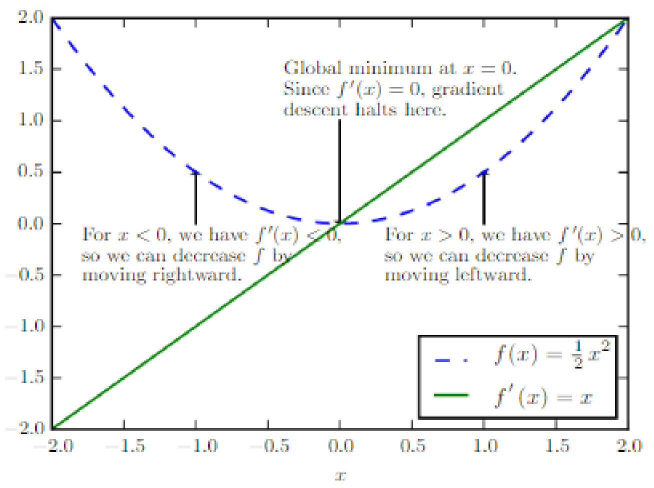


Figure 2.9: An illustration of the gradient descent technique that utilizes the derivatives of the function in order to reach the global minimum [5].

In order to eliminate drawbacks of the vanilla gradient descent, stochastic gradient descent method appears. Instead of updating parameters with respect to the whole training set, it selects one training example, $\boldsymbol{x}_i$, and its label, y_i, at each update:

$$\boldsymbol{\theta} = \boldsymbol{\theta} - \eta \, \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}; \boldsymbol{x}_i; y_i) \tag{2.47}$$

Stochastic gradient descent method removes a redundant computation for large datasets by updating parameters for only a single training instance. Hence, the training is much faster and applicable to online problems. Furthermore, although it suffers from a fluctuation of the loss function due to the variance of the dataset, the researches show that it follows same behavior with a batch gradient descent when the learning rate is decreased slowly during the training [61].

In order to reduce a variance of the parameter updates, mini-batch gradient descent algorithm that is the most popular gradient descent technique is proposed. It performs an update for each mini-batch which constructs N training example:

$$\boldsymbol{\theta} = \boldsymbol{\theta} - \eta \, \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}; \boldsymbol{x}_{i:i+N}; \boldsymbol{y}_{i:i+N}) \tag{2.48}$$

This technique not only reducing a variance of the parameter updates for more stable convergence but also allowing matrix optimizations that could be done in GPU. It induces us to be able to utilize popular deep learning libraries such as Tensorflow [62], PyTorch [63], Caffe [64], etc.

Despite this benefits of the vanilla mini-batch gradient descent, it does not guarantee a good convergence. The reasons for that could be itemized as follows:

- Selecting proper learning rate could not be easy. If the learning rate is too high, the model may overshoot the local minimum whereas a very small learning rate leads to painfully slow convergence.
- Learning rate schedules that are usually exploited to decrease learning rate according to pre-defined schedule to prevent sticking the local minima. However, since it is required to be defined before the training, the schedule may not be suitable for the characteristic of the dataset.

- The same learning rate is applied to all parameter updates. If the data is sparse and it contains dissimilar examples in different frequencies, it causes the network not to learn rare examples.

- When the loss function is highly non-convex, it is nearly impossible to avoid getting stuck in the suboptimal local minima by utilizing vanilla mini-batch gradient descent. Dauphin et. al. [65] claim that since the gradient of the objective function at saddle point is nearly zero in all directions, it is very hard to escape from saddle points for vanilla gradient descent techniques.

There are many advanced gradient descent algorithms which are the extensions of the mini-batch gradient descent to overcome the aforementioned challenges. For further reading, Ruder [61] widely overviews existing advanced gradient descent techniques which are highly exploited in the deep learning area. In this scope of the thesis, adaptive moment estimation (Adam) [66] which is the gradient descent method that calculates adaptive learning rates for each parameter, is utilized since Adam works better in practice while comparing with the other adaptive learning methods.

2.2.4 Backpropagation

In the training phase, after a forward propagation, a scalar cost, $J(\boldsymbol{\theta})$ is obtained. Then, it is required to backpropagate this cost to trainable parameters which belong current and previous layers. The backpropagation is a method that computes gradients of the training parameters according to the objective function through the recursive application of the chain rule. As discussed in Section 2.2.3, the parameters of the network is updated with respect to $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$.

Assume that the objective function which is desired to learn is the multiple composed function, $f(x, y, z) = 5z(2x + 3y)$. Although the gradient of this function is easy to calculate, in order to understand the intuition behind the backpropagation concept, it may be divided into two functions: $q = 2x + 3y$ and $f = 5zq$. The derivatives of these two expressions with respect to their inputs are given below:

$$\frac{\partial f}{\partial q} = 5z, \quad \frac{\partial f}{\partial z} = 5q \tag{2.49}$$

$$\frac{\partial q}{\partial x} = 2, \quad \frac{\partial q}{\partial y} = 3 \tag{2.50}$$

Since the derivatives of the intermediate values are not interested, the value of $\frac{\partial f}{\partial q}$ is not beneficial alone. Instead, the gradient of f with respect to its inputs, x, y, z is significant. $\frac{\partial f}{\partial x}$ and $\frac{\partial f}{\partial y}$ could be computed by utilizing chain rule:

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial x} = 10z \tag{2.51}$$

$$\frac{\partial f}{\partial y} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial y} = 15z \tag{2.52}$$

At the end, the gradient in the variables, $\frac{\partial f}{\partial x}$, $\frac{\partial f}{\partial y}$ and $\frac{\partial f}{\partial z}$ are obtained. They indicate the sensitivity of the variables x, y and z on the function, f. This simple backpropagation example is illustrated in Figure 2.10 where input variables x, y and z are valued to 2, -2 and 1, respectively.

2.2.5 Convolutional Neural Networks

Convolutional neural networks (CNN) are a special type of the neural network that process data which has grid-like topology [67]. It is very similar to deep feedforward networks which are described in Section 2.2.1. CNN also consists of neurons that have weights and biases which are to be learned. Each neuron takes some inputs, performs a dot product and usually applies non-linearity. Besides, there is a still differentiable loss function on the last layer.

The deep feedforward networks do not scale well to images due to their fully connected structure. Assume that the size of images is 400x400x3 in the dataset. It means that there should be 480000 neurons in the first layer and 480000xm weights

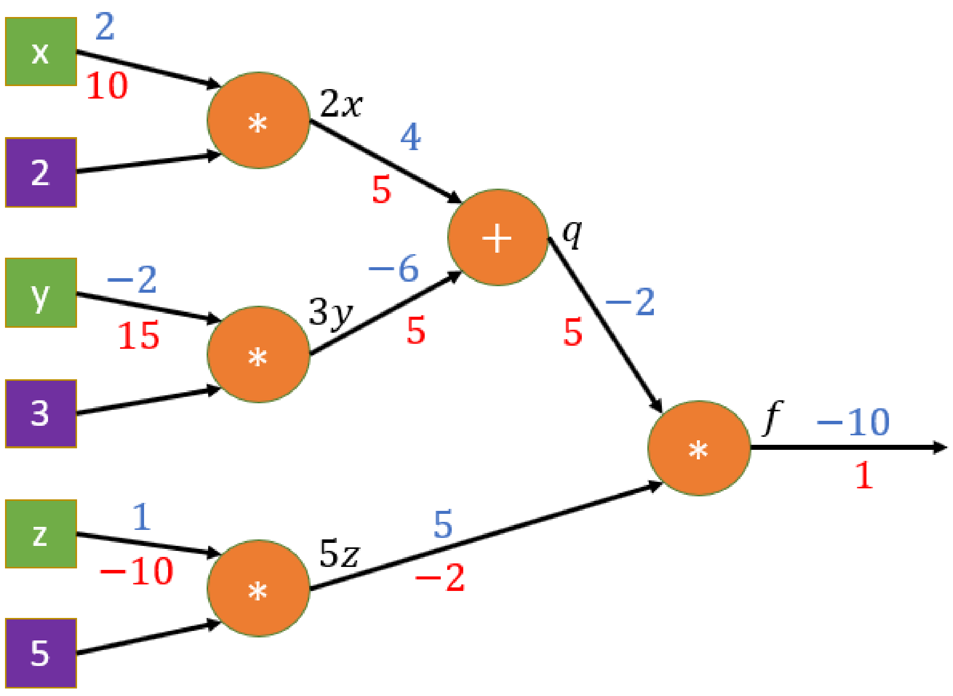
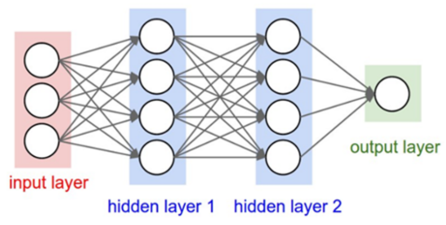
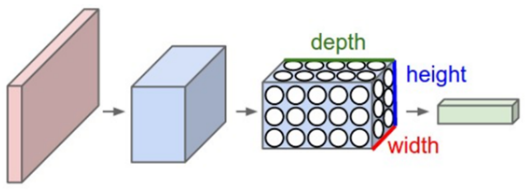


Figure 2.10: A visualization of the forward pass and backpropagation for $f(x, y, z) = 5z(2x+3y)$ expression where $x = 2$, $y = -2$ and $z = 1$. The computed values in forward propagation are indicated in blue whereas the gradients with respect to trainable variables which are calculated by recursively applying chain rule in backpropagation are denoted in red. (Best viewed in color.)

to compute neurons in the first hidden layer where m denotes the number of hidden units in the first hidden layer. Moreover, while the number of hidden layers increases, the parameters size of the neural network increases dramatically. Hence, the problem becomes intractable. Furthermore, the huge number of parameters may lead the network to memorize the dataset quickly instead of learning.



(a)



(b)

Figure 2.11: (a) A conventional 3-layer deep feedforward neural network (b) The layers of CNN consists of neurons which has three dimensions (width, height and depth) as illustrated in one of the layers. The input layer is indicated in red. It has the same size as input images. Hence, its width and height are the dimensions of the images and its depth is three (red, green and blue channels) [4]. (Best viewed in color.)

CNN utilizes the advantage of the inputs that consists of images which have the grid-like topology in a more sensible way. Whereas the neurons of the feedforward neural networks are 1-D, the layers of the CNN are composed of neurons that are arranged in 3 dimensions: width, height and depth. For instance, CIFAR-10 [68] dataset contains images whose size are 32x32x3 (width, height and depth, respectively). As seen in Figure 2.11, the neurons in the current layer are connected to a small field of the next layer instead of fully connecting. Furthermore, the size of the neuron in the final layer is 1x1x10 since there are 10 classes in CIFAR-10 dataset.

A simple CNN consists of a sequence of layers, each layer maps one volume of activations to another via applying differentiable function. There are three main types of layers to construct CNN architectures: convolutional, pooling and fully-connected layer. A simple CNN architecture is visualized in Figure 2.12. A series of the convolutional and pooling operations are performed in CNN architectures. After each convolutional layer, rectified linear units (ReLu) which are non-linear function, max$(x, 0)$, are applied as an activation function. Fully connected layers are utilized after last pooling layer to gather all the information. In the last layer, assuming that the problem is a multi-classification problem, the softmax loss function is exploited.

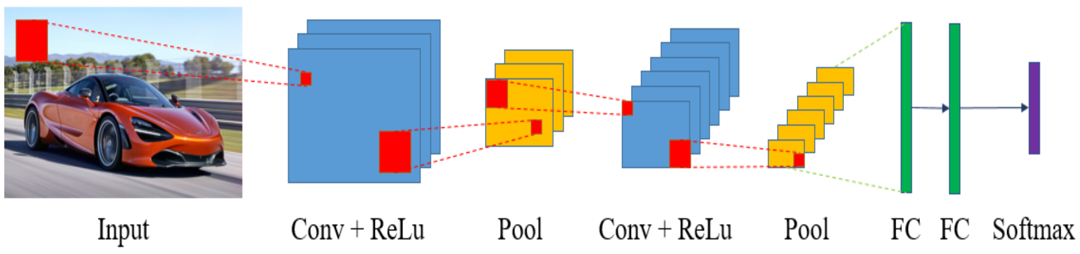


Figure 2.12: A simple CNN architecture example. Conv, Pool and FC denote convolutional, pooling and fully connected layers, respectively. (Best viewed in color.)

The most significant building block of the CNN is the convolutional layer. Convolution is a mathematical operation that provides merging two sets of information. In CNN, the convolution is performed on the input data by utilizing a convolution filter which is also known as a convolution kernel in order to construct a feature map.

A convolution filter is operated by sliding the kernel over the input. At each location, a element-wise matrix multiplication is applied and the result is summed onto the feature map. The example of the convolution operation with the 3x3 kernel is illustrated in Figure 2.13. The size of the kernel is also called as a receptive field where the convolution is performed.

Although this example is useful to understand the convolutional layer, 3-D convolution filters are utilized in real applications instead of 2-D since the input image usually has multiple channels (red, green and blue). Besides, multiple different convolutions are applied on an input to obtain distinct feature maps. Then, these feature maps are stacked to produce the final output of the convolution layer.

Pooling layers usually follow convolutional layers in CNN to reduce the dimensionality. It provides decreasing the number of parameters and preventing from overfitting by downsampling each feature map independently. During pooling, width and height of the inputs are reduced whereas the depth is preserved.

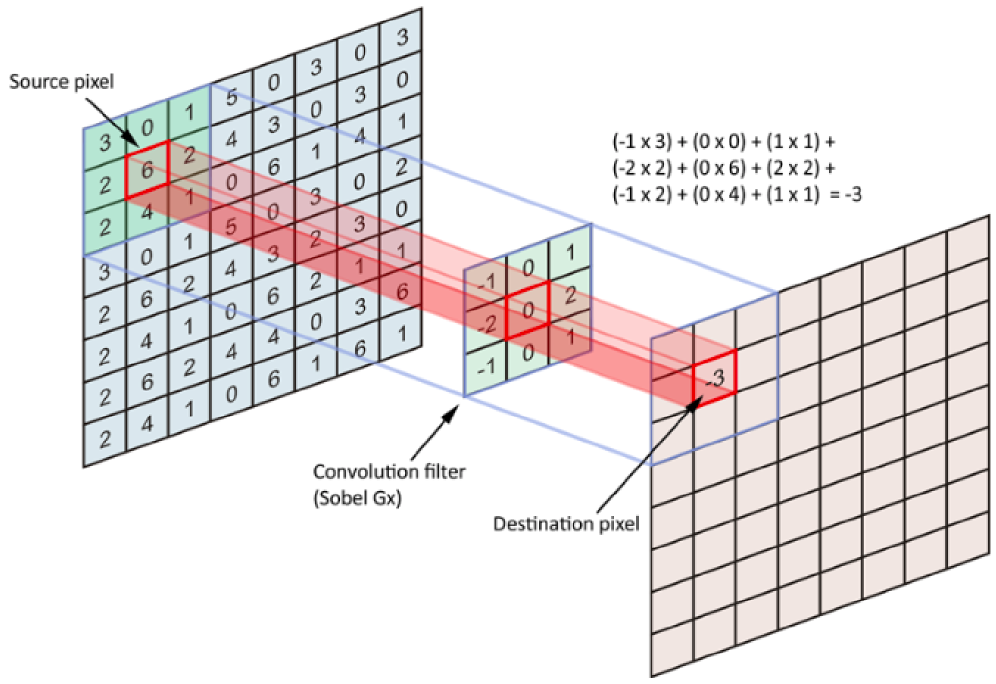

Source pixel
(-1 x 3) + (0 x 0) + (1 x 1) +
(-2 x 2) + (0 x 6) + (2 x 2) +
(-1 x 2) + (0 x 4) + (1 x 1) = -3
Convolution filter
(Sobel Gx)
Destination pixel

Figure 2.13: The 3x3 convolution kernel is applied by sliding over the input to produce the neurons in the next layer [6]. (Best viewed in color.)

The most popular pooling technique is max pooling that returns the maximum value in the pooling window. Unlike the convolutional filter, pooling has no parameter. It is required to define the window size and stride which describes how much to move the pooling window at each step. If the window size and stride are selected as 2x2 and 2, respectively, the pooling operation equals downsampling by 2 as seen in Figure 2.14.

2.2.6 Recurrent Neural Networks

Recurrent neural networks (RNN) are a robust and powerful type of the deep neural networks for processing sequential data. Unlike CNN, RNN could be able to condition the model on all previous inputs in the sequential data.

RNN architecture is illustrated in Figure 2.15 where the prediction of the time instant t is performed. Similar to CNN, each layer contains neurons, each of which applying linear operation on its inputs followed by a non-linear activation function which is $\tanh(.)$ in this case. The hidden units, $\boldsymbol{h}_t$, and the output probability distribution, $\hat{\boldsymbol{y}}_t$,

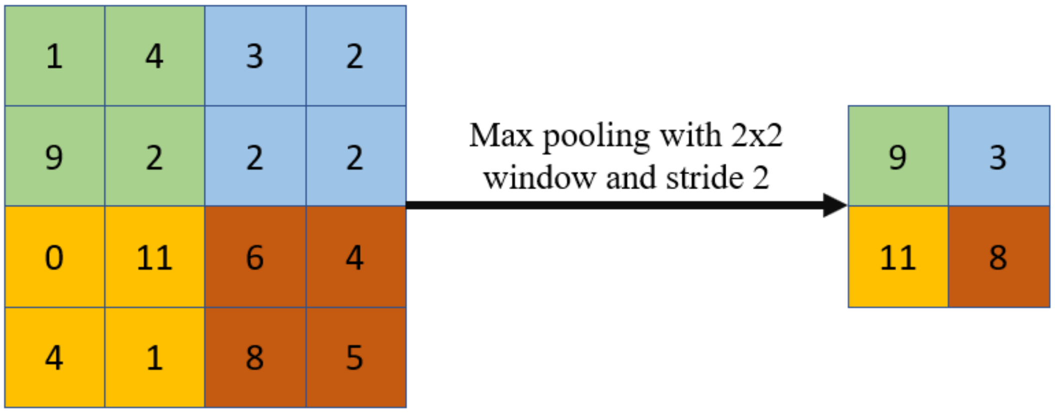


Figure 2.14: Max pooling utilizing 2x2 window and stride 2. Each color indicates a distinct window. (Best viewed in color.)

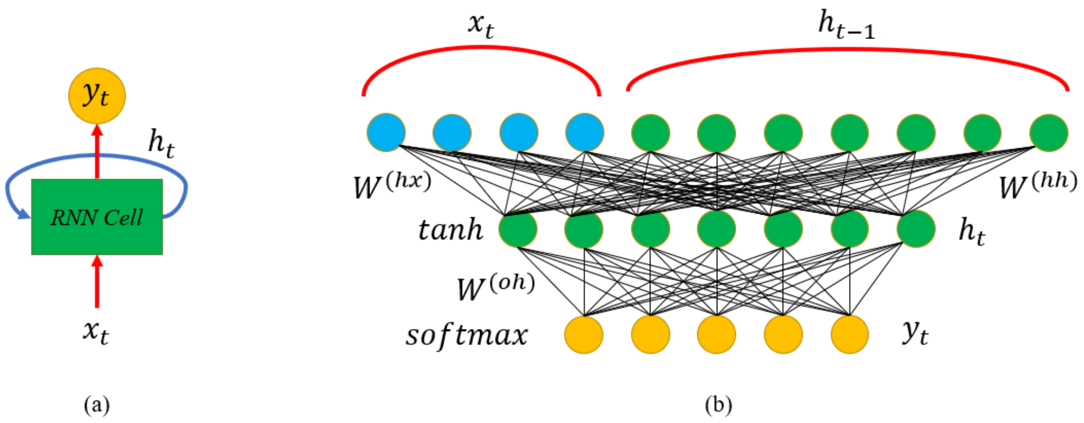


Figure 2.15: (a) The block of the RNN cell. (b) The inside of the RNN cell at a time instant, t. (Best viewed in color.)

at the time instant t are calculated as:

$$\boldsymbol{h}_t = \tanh(\boldsymbol{W}^{hh}\boldsymbol{h}_{t-1} + \boldsymbol{W}^{hx}\boldsymbol{x}_t) \tag{2.53}$$

$$\hat{\boldsymbol{y}}_t = softmax(\boldsymbol{W}^{oh}\boldsymbol{h}_t) \tag{2.54}$$

The parameters of the Equations 2.53 and 2.54 could be described as:

- $\boldsymbol{x}_t \in \mathbb{R}^d$: input vector at time t which has d features.
- $\boldsymbol{W}^{hx} \in \mathbb{R}^{nxd}$: weights matrix from the input layer at the current time-step, $\boldsymbol{x}_t$, to the hidden layer at the current time-step, $\boldsymbol{h}_t$, which is exploited to condition the input vector where n is the number of the hidden neurons.
- $\boldsymbol{W}^{hh} \in \mathbb{R}^{nxn}$: weights matrix from the hidden layer at the previous time-step, $\boldsymbol{h}_{t-1}$, to the hidden layer at the current time-step, $\boldsymbol{h}_t$, which is exploited to condition the hidden vector at time $t-1$.
- $\boldsymbol{h}_{t-1} \in \mathbb{R}^m$: the hidden vector which is produced by applying the non-linear function at the previous time-step.
- $\boldsymbol{h}_0 \in \mathbb{R}^m$: the initialized hidden vector at time, $t = 0$.
- $\boldsymbol{W}^{oh} \in \mathbb{R}^{cxn}$: weights matrix from the hidden layer at the current time-step to the output layer, $\boldsymbol{o}_t$, which is utilized to obtain the output probability distribution at time-step t, $\boldsymbol{y}_t \in \mathbb{R}^c$. If the problem is multiple classification problem, c denotes the number of the classes. However, in scope of this thesis, it is the size of the embedding vector.

In the learning procedure of the RNN is completed by applying backpropagation through time (BPTT) technique which backpropagates error on an unrolled RNN which is visualized in Figure 2.16. By utilizing BPTT, the error is backpropagated from the last to the first time instance while unrolling RNN for all time instances. The rest of the process is the same with the backpropagation concept which is described

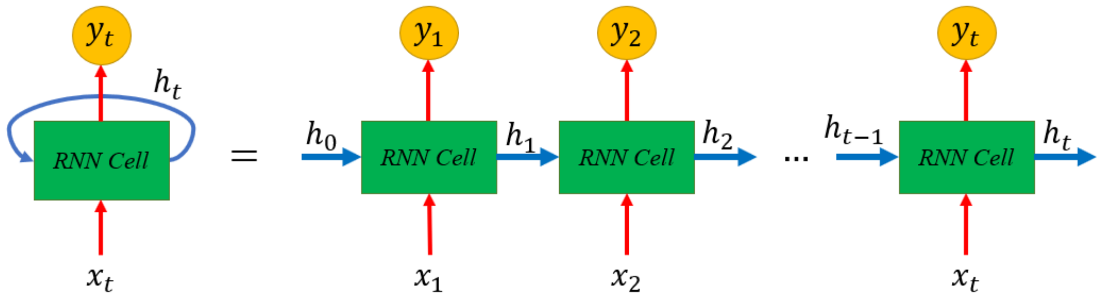


Figure 2.16: The illustration of the unrolled RNN. (Best viewed in color.)

in Section 2.2.4. It is worth to mention that, if the data is large in time, BPTT may be computationally expensive.

Until now, RNN could estimate the output probability distribution by conditioning the model on the previous inputs. It is possible to predict the output based on future inputs by giving inputs in reverse order. Schuster et al. [69] presented bi-directional recurrent neural networks which have two hidden layers at each time-step t, one for the left-to-right propagation whereas the other one is the right to left propagation. The output probability distribution is estimated through concatenating the results of these hidden layers.

Figure 2.17 indicates the deep bi-directional RNN with three hidden layers. In the hidden layers of the bi-directional RNN architecture, each hidden unit receives two inputs which are constructed from left-to-right and right-to-left hidden units at previous time-step in addition to the inputs from the previous layer. Therefore, bi-directional RNN could be formulated in the following equations:

$$\overrightarrow{\boldsymbol{h}}_t^{(i)} = f(\overrightarrow{\boldsymbol{W}}^{(i)}\boldsymbol{h}_t^{(i-1)} + \overrightarrow{\boldsymbol{U}}^{(i)}\overrightarrow{\boldsymbol{h}}_{t-1}^{(i)} + \overrightarrow{\boldsymbol{b}}^{(i)}) \tag{2.55}$$

$$\overleftarrow{\boldsymbol{h}}_t^{(i)} = f(\overleftarrow{\boldsymbol{W}}^{(i)}\boldsymbol{h}_t^{(i-1)} + \overleftarrow{\boldsymbol{U}}^{(i)}\overleftarrow{\boldsymbol{h}}_{t+1}^{(i)} + \overleftarrow{\boldsymbol{b}}^{(i)}) \tag{2.56}$$

$$\hat{\boldsymbol{y}}_t = g(\boldsymbol{V}[\overrightarrow{\boldsymbol{h}}_t^{(o)}; \overleftarrow{\boldsymbol{h}}_t^{(o)}]) \tag{2.57}$$

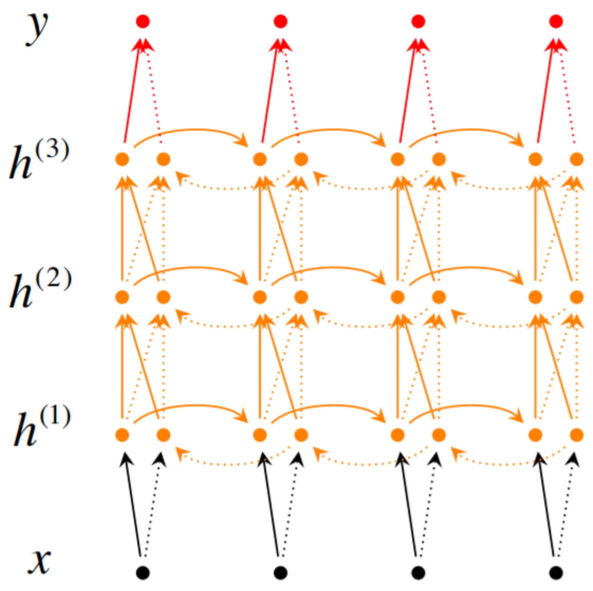


Figure 2.17: The deep bi-directional RNN that consists of three hidden layers [7]. (Best viewed in color.)

where right and left arrows over symbols denote left-to-right RNN and right-to-left RNN, respectively. Superscripts indicate that the parameters belong to which layer. $\boldsymbol{W}$, $\boldsymbol{U}$ and $\boldsymbol{V}$ stands for weights matrices. $\boldsymbol{b}$ denotes a bias vector. $f(.)$ and $g(.)$ are non-linear functions. $[.;.]$ operation means the concatenation along the last axis.

In the theory, a vanilla RNN which is mentioned so far could capture the correlation between inputs, regardless of the number of the time-steps. However, in practice, RNN could grasp only short-term dependencies owing to vanishing and exploding gradient problems. Since there are many multiplications with the same weights matrices in the backpropagation phase of the RNN, if one of the weights matrices is so small, the gradient could be diminished in the more previous layers. This situation is called the vanishing gradient problem. Furthermore, when one of them is very large, the gradients may become infinite in the far-away layers; this issue is called the exploding gradient problem.

In order to remove the exploding gradients problem, the simple heuristic approach that clips gradients over a predefined small value could be utilized. Whenever the value of any gradient exceeds the threshold value, it becomes this threshold value.

There are two common techniques to solve the vanishing gradient problem in RNN. First of all, the weights matrices between hidden layers could be initialized as an identity matrix instead of the matrix whose elements are small. The other technique is to use ReLu, max$(0, x)$, as an activation function instead of tanh and sigmoid functions since the derivative of the ReLu is 0 or 1.

Although the solutions which are mentioned above to capture the long-term dependencies, RNN is still hard to train for this purpose. Two fancy extensions of the RNN cell, gated recurrent units (GRU) [55] and long short-term memories (LSTM) [70], are proposed to address this issue.

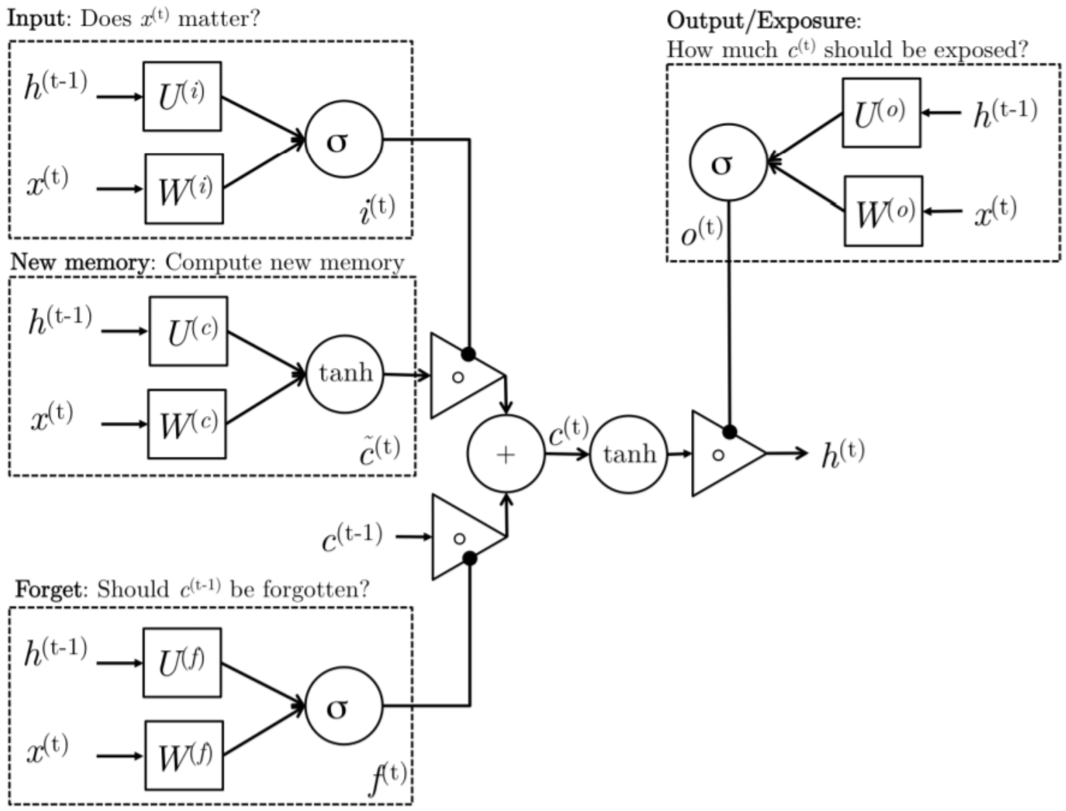


Figure 2.18: The detailed illustration of the LSTM cell [7].

LSTM is designed to capture long-term dependencies by having more permanent memory. Before explaining the intuition behind LSTM, its mathematical expression is described as below:

$$\boldsymbol{i}_t = \sigma(\boldsymbol{W}^{(i)}\boldsymbol{x}_t + \boldsymbol{U}^{(i)}\boldsymbol{h}_{t-1}) \qquad \text{(input gate)} \tag{2.58}$$

$$\boldsymbol{f}_t = \sigma(\boldsymbol{W}^{(f)}\boldsymbol{x}_t + \boldsymbol{U}^{(f)}\boldsymbol{h}_{t-1}) \qquad \text{(forget gate)} \tag{2.59}$$

$$\boldsymbol{o}_t = \sigma(\boldsymbol{W}^{(o)}\boldsymbol{x}_t + \boldsymbol{U}^{(o)}\boldsymbol{h}_{t-1}) \qquad \text{(output gate)} \tag{2.60}$$

$$\hat{\boldsymbol{c}}_t = \tanh(\boldsymbol{W}^{(c)}\boldsymbol{x}_t + \boldsymbol{U}^{(c)}\boldsymbol{h}_{t-1}) \qquad \text{(new memory cell)} \tag{2.61}$$

$$\boldsymbol{c}_t = \boldsymbol{f}_t \circ \boldsymbol{c}_{t-1} + \boldsymbol{i}_t \circ \hat{\boldsymbol{c}}_t \qquad \text{(final memory cell)} \tag{2.62}$$

$$\boldsymbol{h}_t = \boldsymbol{o}_t \circ \tanh(\boldsymbol{c}_t) \qquad \text{(current hidden state)} \tag{2.63}$$

The intuition behind the LSTM cell could be understood in the following stages which are seen in Figure 2.18:

1. **New memory generation:** In this stage, a new memory cell, $\hat{\boldsymbol{c}}_t$, is produced by merging the input at current time-step, $\boldsymbol{x}_t$, with the previous hidden state, $\boldsymbol{h}_{t-1}$, to describe the correlation between $\boldsymbol{x}_t$ and $\boldsymbol{h}_{t-1}$.
2. **Input gate:** This gate utilizes $\boldsymbol{x}_t$ and $\boldsymbol{h}_{t-1}$ in order to decide how much of $\boldsymbol{x}_t$ is significant to be preserved.
3. **Forget gate:** It determines whether the previous memory cell, $\boldsymbol{c}_{t-1}$, is important to compute the current memory cell, $\boldsymbol{c}_t$, by looking the correlation between $\boldsymbol{x}_t$ and $\boldsymbol{h}_{t-1}$.
4. **Final memory generation:** At first, this stage forgets the past memory, $\boldsymbol{c}_{t-1}$, by listening the forget gate, $\boldsymbol{f}_t$. Then, it pays attention to the input gate, $\boldsymbol{f}_t$, to understand the required information of the new memory cell. Finally, it combines two pieces of the information by adding operation to generate the final memory, $\boldsymbol{c}_t$.
5. **Output gate:** This gate, $\boldsymbol{o}_t$, is utilized to determine required information of the final memory, $\boldsymbol{c}_t$, to be stored in the hidden state, $\boldsymbol{h}_t$.

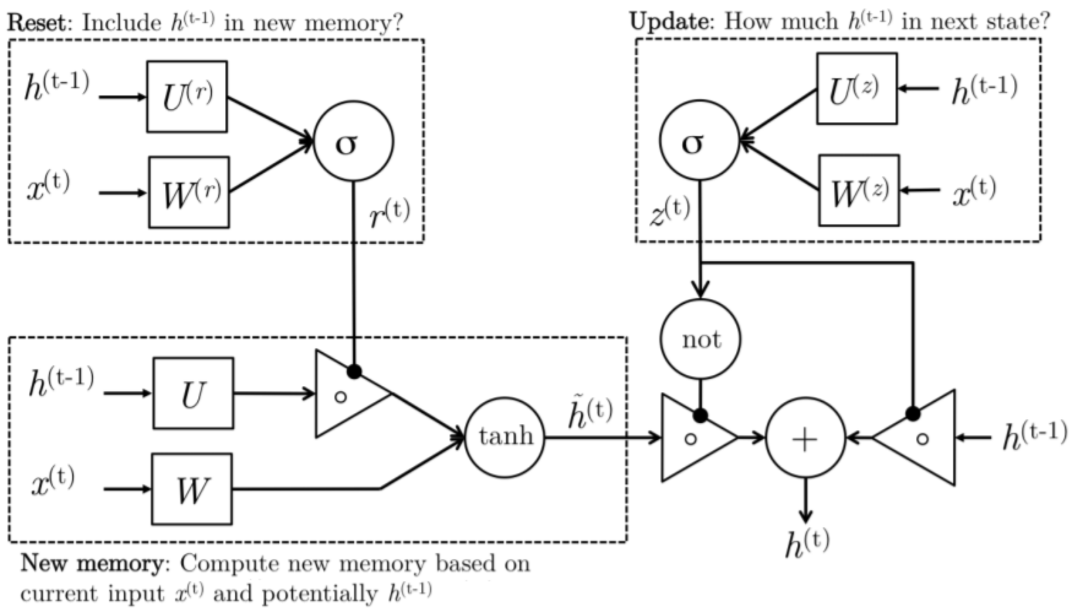

Reset: Include $h^{(t-1)}$ in new memory?
$h^{(t-1)}$
$U^{(r)}$
$x^{(t)}$
$W^{(r)}$
σ
$r^{(t)}$
Update: How much $h^{(t-1)}$ in next state?
$U^{(z)}$
$h^{(t-1)}$
σ
$W^{(z)}$
$x^{(t)}$
$z^{(t)}$
not
$h^{(t-1)}$
U
$x^{(t)}$
W
tanh
$\tilde{h}^{(t)}$
+
$h^{(t-1)}$
$h^{(t)}$
New memory: Compute new memory based on current input $x^{(t)}$ and potentially $h^{(t-1)}$

Figure 2.19: The detailed visualization of the GRU cell [7].

Although the motivation of designing GRU is same with LSTM, GRU is computationally cheaper than LSTM since it has fewer parameters. Before mentioning the idea behind GRU, it could be formulated as:

$$\boldsymbol{z}_t = \sigma(\boldsymbol{W}^{(z)}\boldsymbol{x}_t + \boldsymbol{U}^{(z)}\boldsymbol{h}_{t-1}) \qquad \text{(update gate)} \tag{2.64}$$

$$\boldsymbol{r}_t = \sigma(\boldsymbol{W}^{(r)}\boldsymbol{x}_t + \boldsymbol{U}^{(r)}\boldsymbol{h}_{t-1}) \qquad \text{(reset gate)} \tag{2.65}$$

$$\hat{\boldsymbol{h}}_t = \tanh(\boldsymbol{W}\boldsymbol{x}_t + \boldsymbol{r}_t \circ \boldsymbol{U}\boldsymbol{h}_{t-1}) \qquad \text{(new memory)} \tag{2.66}$$

$$\boldsymbol{h}_t = (1 - \boldsymbol{z}_t) \circ \hat{\boldsymbol{h}}_t + \boldsymbol{z}_t \circ \boldsymbol{h}_{t-1} \qquad \text{(hidden state)} \tag{2.67}$$

The above equations could be considered as four main stages of the GRU cell as illustrated in Figure 2.19:

1. **New memory generation:** In this stage, a new memory cell, $\hat{\boldsymbol{h}}_t$, is generated

by combining the input at current time-step, $\boldsymbol{x}_t$, with the previous hidden state, $\boldsymbol{h}_{t-1}$, to summarize $\boldsymbol{x}_t$ according to $\boldsymbol{h}_{t-1}$.

2. **Reset gate:** This gate exploits $\boldsymbol{x}_t$ and $\boldsymbol{h}_{t-1}$ in order to decide how important $\boldsymbol{x}_t$ is to summarize $\hat{\boldsymbol{h}}_t$. The reset gate, $\boldsymbol{r}_t$ has power to reduce past hidden states if it decides that $\boldsymbol{h}_{t-1}$ is not significant to compute $\hat{\boldsymbol{h}}_t$.

3. **Update gate:** It determines whether the previous memory cell, $\boldsymbol{h}_{t-1}$, is important to compute the current hidden state, $\boldsymbol{c}_t$, by looking the correlation between $\boldsymbol{x}_t$ and $\boldsymbol{h}_{t-1}$. If update gate, $\boldsymbol{z}_t$, is 1, the current hidden state is same with the previous hidden state. Conversely, if it is 0, $\boldsymbol{h}_t$ become the value of $\hat{\boldsymbol{h}}_t$.

4. **Hidden state:** Finally, the hidden state, $\boldsymbol{h}_t$, is produced by combining $\boldsymbol{h}_{t-1}$ and $\hat{\boldsymbol{h}}_t$ with the advice of $\boldsymbol{z}_t$.

2.2.7 Encoder-Decoder Models with Attention Mechanism

RNN could be utilized to map an input sequence to a fixed-length vector. In the Section 2.2.6, mapping the input and the output which have the same size is described. However, the size of the input and the output could be different in some applications such as machine translation, speech recognition, question answering etc.

The first RNN architectures which are utilized to map a variable-length sequence to another variable-length sequence was introduced by Cho et. al. [56] and shortly later by Sutskever et al. [57] in order to apply the encoder-decoder model to the machine translation problem. This encoder-decoder RNN architecture is illustrated in Figure 2.20 for a machine translation task from English to Turkish.

The intuition behind the encoder-decoder architecture is simple. There are two RNN architectures, encoder and decoder, which have different parameters. At first, the encoder compresses the information of the input sequences, $\boldsymbol{X} = (\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_N)$, into the variable length context vector, $\boldsymbol{c}$. Then, the encoder transmits $\boldsymbol{c}$ which is obviously the final hidden state of the decoder to the encoder. The encoder receives $\boldsymbol{c}$ and utilizes this context vector by assigning it to the initial state of the encoder. After that, the decoder produces the output sequence, $\boldsymbol{Y} = (\boldsymbol{y}_1, \boldsymbol{y}_2, ..., \boldsymbol{y}_M)$. As mentioned before, The lengths of input and output sequence,N and M, could be different. In this architecture, these two encoder and decoder RNNs are trained jointly to maximize the conditional probability, $P(\boldsymbol{y}_1, \boldsymbol{y}_2, ..., \boldsymbol{y}_M | \boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_N)$.

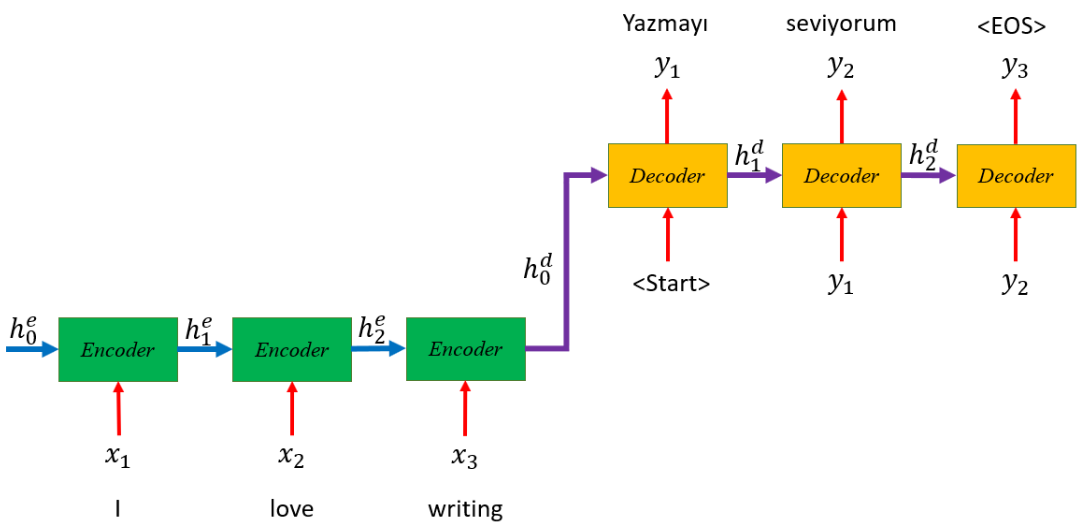


Figure 2.20: An example of the encoder-decoder or sequence-to-sequence RNN model for a machine translation problem from English to Turkish. <Start> and <EOS> are tokens which denote the start and end of the sentence. (Best viewed in color.)

The most important limitation of the encoder-decoder model is that the encoder may not summarize a long sequence since the size of the context vector could be inadequate to represent a long sequence. In order to address this issue, Bahdanau et al. [55] proposed an attention mechanism which learns the relation between elements of the context sequence, $\boldsymbol{c}_t$, and the elements of the output sequence, $\boldsymbol{y}_t$. Where encoder and decoder hidden states at time-step t are $\boldsymbol{h}_t$ and $\boldsymbol{s}_t$, respectively, the attention in encoder-decoder RNN models could be formulated as:

$$\boldsymbol{e}^t = [\boldsymbol{s}_t^T \boldsymbol{h}_1, \boldsymbol{s}_t^T \boldsymbol{h}_2, ..., \boldsymbol{s}_t^T \boldsymbol{h}_N] \in \mathbb{R}^N \tag{2.68}$$

$$\alpha^t = \text{softmax}(\boldsymbol{e}^t) \in \mathbb{R}^N \tag{2.69}$$

$$\boldsymbol{a}_t = \sum_{i=1}^{N} \alpha_i^t(\boldsymbol{h}_i) \in \mathbb{R}^m \tag{2.70}$$

As seen equations above, at first, attention scores, $\boldsymbol{e}^t$, are computed for input hidden vectors at each time-step to determine the association with the focused output at the time-step t, $\boldsymbol{y}_t$, by applying dot product where M describes the length of the input sequence. Then, softmax function is applied to these attention scores in order to obtain the attention distribution, α^t, for the current decoder time-step t. After that, attention output, $\boldsymbol{a}_t$, is produced by taking a weighted sum of the encoder hidden states where m is the number of hidden units in the encoder. Finally, the output probability distribution, $\hat{\boldsymbol{y}}_t$, is computed while its input is generated by concatenating attention output with decoder hidden states along the last dimension.

Figure 2.21 illustrates these mentioned steps. On the decoder time-step where the output probability distribution is produced in that example, the network is mostly focused on *"pauvres"* input word while predicting *"poor"* output word.

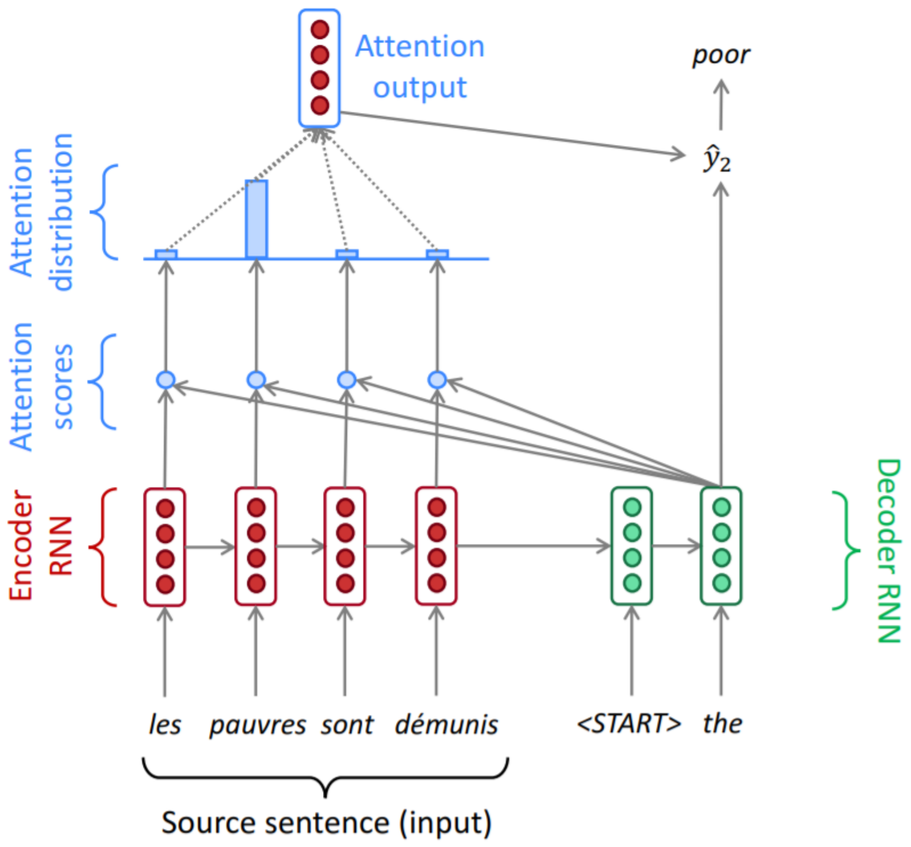


Figure 2.21: An illustration of the encoder-decoder RNN architecture with an attention mechanism while predicting the output at the time-step, $t = 2$, [7]. (Best viewed in color.)

CHAPTER 3

METRIC LEARNING USING DEEP LEARNED REPRESENTATIONS

In this chapter of the thesis, a novel deep metric learning loss function and a sampling method are presented. Following the explanation of the motivation behind the novel deep metric learning approach, the general framework of deep metric learning based on CNN is described. Then, each part of the algorithm is introduced in details under three main sections. Finally, the efficacy of the proposed approach is validated with quantitative and qualitative experimental results.

3.1 Motivation

The metric learning methods based on deep neural networks are detailedly examined in terms of their advantages and disadvantages in Section 2.1.2. One of the most major issue in existing methods is to select proper negative examples while constructing mini-batch.

Triplet embedding approach which is described in Section 2.1.2.2 utilizes mini-batch construction method that associates each positive pair in the mini-batch with the *"semi-hard"* negative examples. However, the size of the mini-batches is required to be very large in order to ensure that each mini-batch has adequate hard negatives. In N-pair embedding algorithm mentioned in Section 2.1.2.4, this mini-batch construction idea is improved by selecting N different positive pairs. The positive examples of each pair should violate triplet constraint with respect to the anchors that belong to the dissimilar classes. This mini-batch creation approach is called N-pair sampling. Although the results on selected datasets are significantly improved by utilizing this sampling algorithm [30], there are two key limitations of N-pair embedding:

1. Each anchor has one positive example and $N-1$ negative examples in order to ensure enough number of hard negatives due to the computational constraints in GPU. However, it causes reducing the representation power of the loss function since the embedding representations of the dissimilar examples are mapped to far away points in the embedding space from the embedding representations of the anchors by considering only one positive example.

2. Despite increasing the number of negative examples for each anchor, selecting representative hard negatives is still not guaranteed since the number of examples in the mini-batch, $2N$, could not be large as much as required.

Both of these deficiencies come out due to a lack of the number of representative hard negatives. As it is expected, this is an old research area and there is a conventional solution, originally called *bootstrapping* which is now referred to as *hard negative mining*. Sung [71] introduces bootstrapping method for training face detection models. This approach constructs the set of dissimilar examples from wrongly classified negative examples by gradually growing or bootstrapping. Hence, two training sets, current and bootstrapped, are utilized to update model parameters in an iterative manner.

The common motivation of hard negative mining algorithms is selecting easier samples as inputs in the training phase may lead the converge rate for model parameters to be reduced since easier negatives usually contribute smaller gradients. Therefore, emphasizing hard negative samples during training may accelerate the learning process. An example could be defined as hard negative or not by looking its loss during training process [33]. It means that the embeddings of the whole dataset should be computed at each training iteration in order to obtain hard negative samples. Due to this reason, it is too much computationally expensive to be performed in large training sets.

To alleviate aforementioned drawbacks of utilizing N-pair sampling and hard negative mining, a novel local neighborhood sampling approach is proposed. This method constructs mini-batches whose samples are randomly selected example defined as *center*, and examples that belong to its k-nearest neighborhood in the learned embedding space.

The idea behind this proposed sampling method is developed by inspiring from the performance of existing methods [1, 9, 15, 28, 29, 30, 31, 32] in the literature on Recall@K metric that is the proportion of similar examples found in the top-K retrievals. For instance, by following training procedure in [32], whereas the Recall@1 performance of the N-pair embedding method is 66.4%, Recall@100 metric is 92.1% on the Stanford Online Products dataset [15]. It means that hard negatives which are required to be focused for each sample is produced in its k-nearest neighborhood at the end of the training. In the light of these observations, after training with N-pair sampling through the predefined number of iterations, the learning process could be continued as utilizing hard negatives that are produced by introduced local neighborhood sampling.

Since the restrictions of existing sampling methods on the deep metric learning loss functions are reduced owing to proposed sampling method, a novel loss function which is an extended version of $(N+1)$-tuplet loss function described in Section 2.1.2.4, could be introduced. This proposed loss function optimizes to identify P similar examples from N negative examples. It provides enhancing the representation power of the network since the negative examples are kept away by considering multiple similar samples instead of considering only one positive example in N-pair objective function; when $P = 1$, it is equivalent to $(N+1)$-tuplet loss.

3.2 Deep Metric Learning Framework

Let $\boldsymbol{x}_i \in \boldsymbol{X}$ be an input vector which is the element of training set and $y_i \in \{1, 2, ..., C\}$ be a class label of $\boldsymbol{x}_i$ where C denotes the number of different classes in dataset. The aim of the deep metric learning is to learn non-linear function, $f_{\boldsymbol{\theta}}(\cdot)$, such that embedding representations of similar examples are mapped to close points on a manifold and embedding representations of dissimilar examples are mapped to far away points.

Figure 3.1 illustrates the overview scheme of deep metric learning algorithms. At first, mini-batches which have N examples, are constructed by applying a sampling algorithm that selects representative positive and negative samples from the training set, $\boldsymbol{X}$. Then, the non-linear function, $f_{\boldsymbol{\theta}}(\cdot)$, which is CNN described in 2.2.5 takes each of the examples in the mini-batch, $\boldsymbol{x}_i$, as an input and it produces the embedding representation of $\boldsymbol{x}_i$, $f_{\boldsymbol{\theta}}(\boldsymbol{x}_i)$. Note that, by generating each $\boldsymbol{x}_i$, every CNN shares its parameters. In other words, each CNN has the same weights and biases. Finally, after obtaining embedding representations of all samples in the mini-batch, the network is optimized with respect to deep metric learning loss function with the aid of mini-batch gradient descent and backpropagation which are introduced in Section 2.2.3 and 2.2.4, respectively.

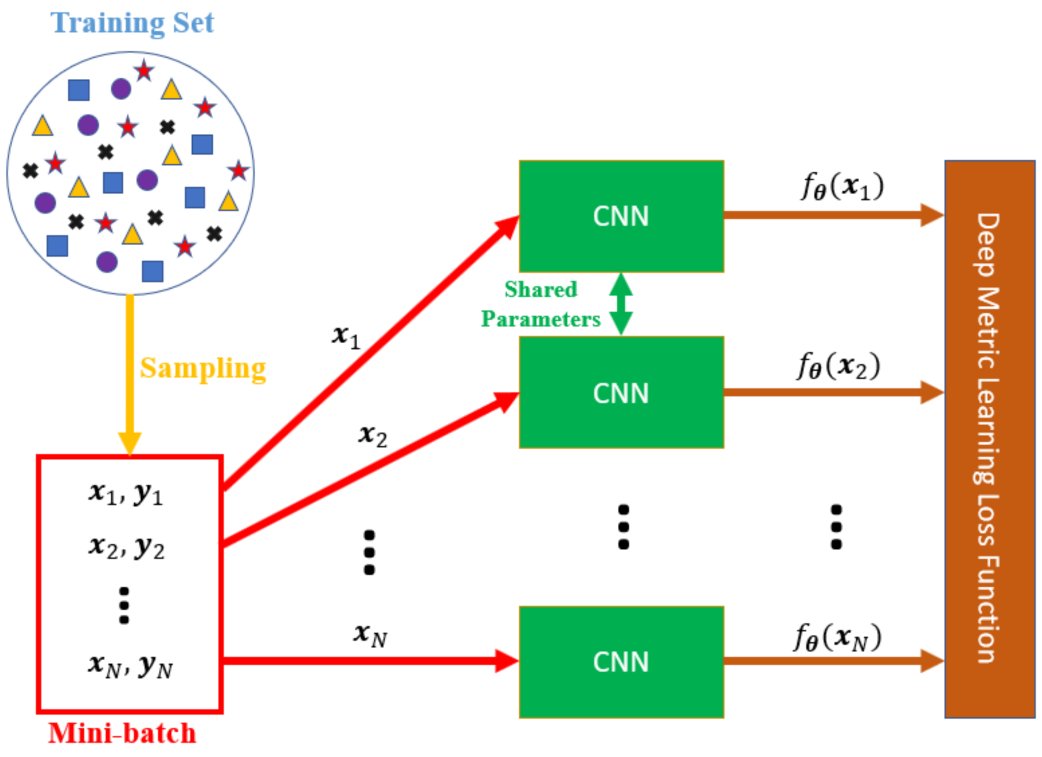


Figure 3.1: The general framework of the deep metric learning algorithms. (Best viewed in color.)

3.3 Proposed Loss Function

A deep metric learning loss function is proposed in order to recognize multiple similar examples from multiple dissimilar samples. $(N{+}P{+}1)$-tuplet of training sample which is visualized in Figure 3.2 could be constructed as $\{\boldsymbol{x}^{anc}, \boldsymbol{x}_i^{pos}, ..., \boldsymbol{x}_P^{pos}, \boldsymbol{x}_j^{neg}, ..., \boldsymbol{x}_N^{neg}\}$ where $\boldsymbol{x}^{anc}$, $\boldsymbol{x}_i^{pos}$ and $\boldsymbol{x}_j^{neg}$ denote an *anchor*, a positive example at index i of $\boldsymbol{x}^{anc}$ and a negative sample at index j of $\boldsymbol{x}^{anc}$. The $(N{+}P{+}1)$-tuplet loss function could be formulated as below:

$$J_{(N+P+1)\text{-tuplet}} = \log(1 + \frac{1}{P}\sum_{i=1}^{P}\sum_{j=1}^{N}\exp\{f_{\boldsymbol{\theta}}(\boldsymbol{x}^{anc})^T f_{\boldsymbol{\theta}}(\boldsymbol{x}_j^{neg}) - f_{\boldsymbol{\theta}}(\boldsymbol{x}^{anc})^T f_{\boldsymbol{\theta}}(\boldsymbol{x}_i^{pos})\}) \tag{3.1}$$

where $f_{\boldsymbol{\theta}}$ stands for the non-linear embedding function whose parameters are $\boldsymbol{\theta}$. Recall that when $P = 1$, it is equivalent to (N+2)-tuplet loss which is renamed from (N+1)-tuplet loss in [30] to provide name consistency with the proposed loss. Furthermore, it is worth the mention that if $P = 1$ and $N = 1$, the loss function is highly similar to the triplet loss.

Since the desired loss described in Equation 3.1 requires all positive and negative examples of the selected anchor, it is impractical to utilize this objective function in large datasets. To alleviate this problem, the sets of positive and negative examples, $\mathbb{P}_k$ and $\mathbb{N}_k$, respectively, could be selected from the k-nearest neighborhood of the anchor, $\mathbb{K}$, which is indicated as subscript c from now. The further reasons of this simplification are argued in Section 3.4. Hence, where $D_{i,j} = f_{\boldsymbol{\theta}}(\boldsymbol{x_i})^T f_{\boldsymbol{\theta}}(\boldsymbol{x_j})$,the loss function could be reformulated as:

$$J_{k\text{-nearest}} = \log(1 + \frac{1}{|\mathbb{P}_k|}\sum_{i\in\mathbb{P}_k}\sum_{j\in\mathbb{N}_k}\exp\{D_{c,j} - D_{c,i}\}) = -\log\frac{\frac{1}{|\mathbb{P}_k|}\sum_{i\in\mathbb{P}_k}\exp\{D_{c,i}\}}{\sum_{k\in\mathbb{K}}\exp\{D_{c,k}\}} \tag{3.2}$$

Equation 3.2 resembles the multi-class softmax loss function where C, y_j and z_i are the number of classes, desired output class and unnormalized log-probabilities, respectively, which is defined as:

$$J(\theta) = -\sum_{i=1}^{C} y_j \log\frac{\exp z_i}{\sum_j \exp z_j} \tag{3.3}$$

In order to train $J_{k\text{-nearest}}$ with the softmax loss for easy implementation since all deep learning libraries have cross entropy with softmax loss function, easy trick could be applied. By summation over $|\mathbb{K}|$ instead of C, $\boldsymbol{y} \in \mathbb{R}^{|\mathbb{K}|}$ could be constructed as $\{\frac{y_1}{\max(1,|\mathbb{P}|)}, \frac{y_2}{\max(1,|\mathbb{P}|)}, \ldots, \frac{y_{|\mathbb{K}|}}{\max(1,|\mathbb{P}|)}\}$ where $y_i = 1$ if $y_i \in \mathbb{P}$ and $y_i = 0$ when $y_i \in \mathbb{N}$.

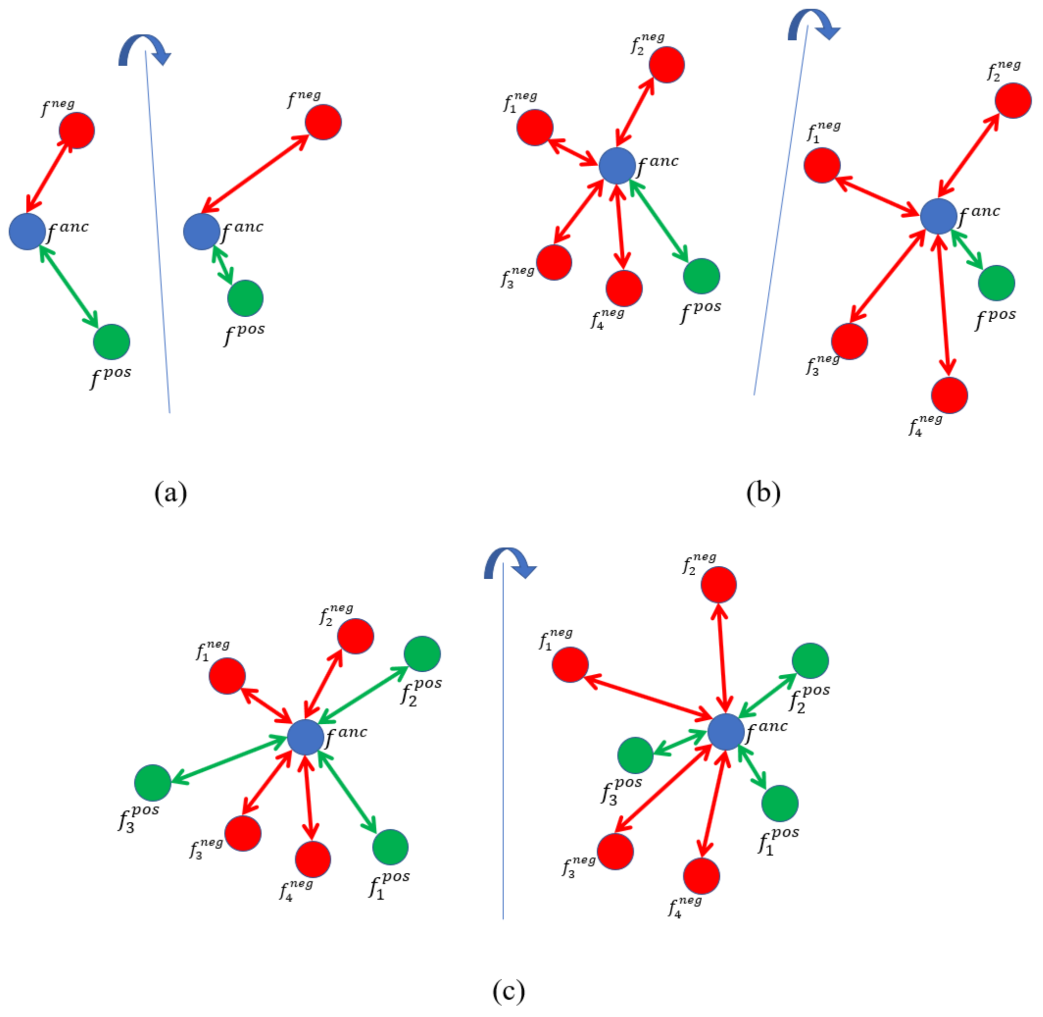


Figure 3.2: Deep metric learning with (a) triplet described in Section 2.1.2.2, (b) (N+2)-tuplet defined in Section 2.1.2.4 and (c) proposed (N+P+1)-tuplet embedding loss functions. While pulling the embedding vector of positive example, f^{pos}, to the point of an anchor in the embedding space, f^{anc}, (N+2)-tuplet objective function keeps away embedding vectors of N negative example, f_i^{neg}, from f^{anc} instead of pushing only one f^{neg} like the triplet loss. On the other hand, in proposed (N+P+1)-tuplet loss function, P f_i^{pos} are getting closer to f^{anc} while dragging away N negative examples. (Best viewed in color.)

3.4 Local Neighborhood Sampling

When the mini-batch size is M, M x $(K+1)$ examples are required to be passed through the network where K stands for $|\mathbb{K}|$. It could be intractable for convolutional neural networks that have large parameters since the number of input samples increases in quadratic to M and K.

In order to reduce this computational complexity, a novel batch construct method is proposed. At first, the network is trained by utilizing N-pair sampling [30] with a minor modification to produce k-nearest neighbors of the examples. In original N-pair sampling method which is indicated in Figure 3.3(b), N pair of similar examples are selected from different classes that violates triplet constraint defined in Section 2.1.2.2. One example of each class is described as an anchor and the other positive examples which are dissimilar to an anchor is chosen as negative examples. This batch construction approach constructs mini-batches whose size are N from $2N$ samples whereas triplet construction illustrated in Figure 3.3(a) requires $3N$ examples. In order to provide full mini-batch usage like lifted structured embedding loss explained in Section 2.1.2.3 , each sample in mini-batch is defined as an anchor or a *center*. Instead of taking one of the pairs which have different classes from the center, each dissimilar sample in mini-batch is assigned as a negative of the center.

After constructing mini-batches by utilizing this modified N-pair sampling which is also illustrated in Figure 3.3(c), the deep neural network is trained through predefined iterations. Although training could continue until the parameters of the network converge, the number of iterations is predetermined to maintain the end-to-end training.

Therefore, the trained neural network , $f_{pre-trained}(\cdot)$, is ready to extract hard negatives. The embeddings of samples in the training set is produced only once by applying $f_{pre-trained}(\cdot)$. Assume that $\mathbb{C}$ is the set of centers in the mini-batch and $\mathbb{K}_c$ is constructed from K closest examples to each center at index c in the embedding space that is mapped by $f_{pre-trained}(\cdot)$. In order to minimize error on the mini-batch,

Equation 3.2 becomes:

$$J_{k\text{-nearest-mb}} = \frac{1}{|\mathbb{C}|} \sum_{c \in \mathbb{C}} -\log \frac{\frac{1}{|\mathbb{P}_k|} \sum_{i \in \mathbb{P}_k} \exp\{D_{c,i}\}}{\sum_{k \in \mathbb{K}_c} \exp\{D_{c,k}\}} \tag{3.4}$$

where $\mathbb{P}_k$ is a set of positive examples of the center, c, in $\mathbb{K}_c$. Note that the network gains robustness to outliers by taking positive examples in the k-nearest neighborhood of each sample instead of all similar examples in the training set. It has the same spirit to *self-paced* learning [34] which makes the model more robust to outliers by decreasing weights of hard similar examples in the loss function.

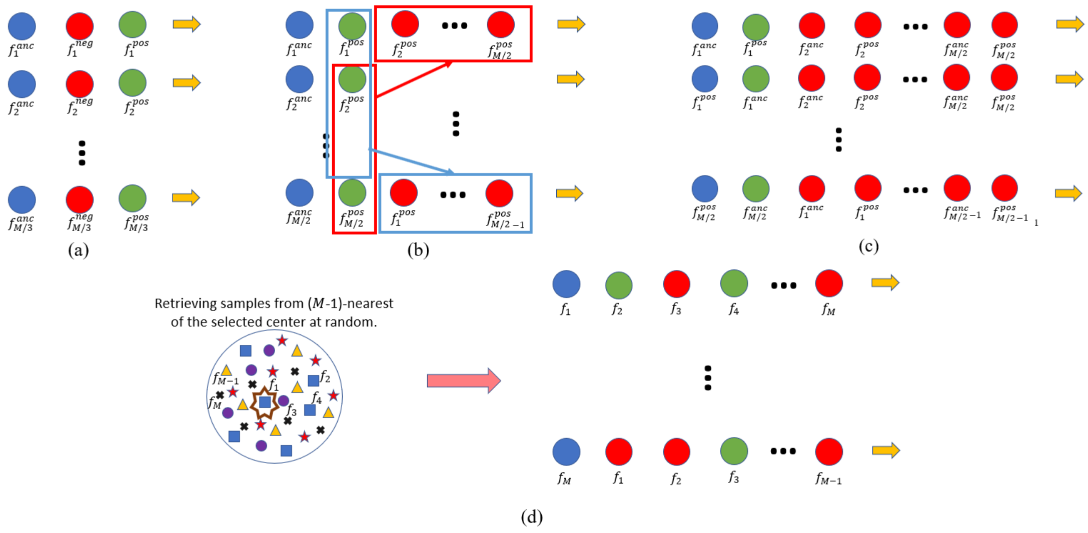

(a)
(b)
(c)
(d)
Retrieving samples from (M-1)-nearest of the selected center at random.

Figure 3.3: (a) triplet (b) N-pair (c) modified N-pair (d) local neighborhood (when $|\mathbb{C}|$=1) sampling methods. In order to construct mini-batch that consists of N queries, Triplet, N-pair, modified N-pair and local neighborhood batch construction approaches require $3N$, $2N$, N and N_p passes to produce embedding vectors, respectively. N_p describes the number of samples which has at least one similar example in the mini-batch. (Best viewed in color.)

To receive full information of each manifold in the mini-batch whose elements are center, c, and its $|\mathbb{K}_c|$ closest samples in the embedding space, $\mathbb{C}$ is constructed by assuming all samples in $\mathbb{K}_c$ is the center. Although this approximation produces correlated mini-batches and the aim of the most machine learning algorithms is constructing independent and identically distributed mini-batches to increase the performance of the gradient descent techniques, existing deep metric learning methods described in Section 2.1.2 want to ensure that there are representative similar and dissimilar examples in the mini-batch. For future research directions, there is a study in [72] that removes the negative effects of mini-batches which consists of correlated samples by applying batch renormalization. Moreover, in order to taking advantage of utilizing more negatives, each example in the mini-batch could be considered as the center of the mini-batch. Hence, Equation 3.4 could be reformulated as:

$$J_{\text{centralized-mb}} = \frac{1}{N_p} \sum_{i=1}^{N} - \log \frac{\frac{1}{|\mathbb{P}_i|} \sum_{j \in \mathbb{P}_i} \exp\{D_{i,j}\}}{\sum_{k \neq i} \exp\{D_{i,k}\}} \tag{3.5}$$

where $\mathbb{P}_i$ is the set that contains positive examples in the mini-batch of the center, i. N stands for the number of examples in the mini-batch whereas N_p denotes the number of samples which has at least one similar example in the mini-batch. Therefore, mini-batch size, M, is equal to N_p. The reason behind dividing summation over a set of centers by N_p is that examples which have not similar samples in the mini-batch contribute zero gradients.

Finally, local neighborhood sampling which is visualized in 3.3(d) could be summarized in the following steps:

1. **Store Embedding Vectors:** pass all training set through the pre-trained network, $f_{pre-trained}(\cdot)$, to extract the embedding vector of each sample, and store each produced embedding vector in $\boldsymbol{Z}_{pre-trained} \in \mathbb{R}^{mxd}$ where m and n denote the number of examples and embedding size, respectively.

2. **Select Center:** select one sample at random. Then, retrieve pre-defined $|\mathbb{K}|$ closest examples to the center according to $\boldsymbol{Z}_{pre-trained}$.

3. **Check Positive and Negative Examples:** check sample labels in $\mathbb{K}$. If there are at least one similar and one dissimilar example to the center, construct a temporary set, $\mathbb{S}_{temp}$, from the center and retrieved samples. If not, return step 2.

4. **Check Correlation:** check whether one of the elements in $\mathbb{S}_{temp}$ is already selected or not. If it is, return step 2.

5. **Finalize Batch Construction:** extend the set, $\mathbb{S}$, which consists of examples in mini-batch with the samples in $\mathbb{S}_{temp}$. Go to step 2 until $|\mathbb{S}|$ is equal to N which denote the required number of examples in mini-batch.

3.5 Regularization of Embedding Vectors

Since local neighborhood sampling constructs mini-batches by utilizing neighborhood information in $\boldsymbol{Z}_{pre-trained}$ which is calculated only once to reduce computational complexity instead of each iteration passing all training samples through the network, the embedding representations of negative samples may be mapped into undesired far points from the center. In order to prevent this challenge, L^2 norm of the difference between produced embedding vectors and precomputed embedding vectors could be regularized to be small:

$$J_{\text{reg-pre}} = \frac{1}{N}\sum_{i=1}^{N}\|f_{\boldsymbol{\theta}}(\boldsymbol{x}_i) - f_{pre-trained}(\boldsymbol{x}_i)\|_2 \tag{3.6}$$

where $f_{\boldsymbol{\theta}}(\cdot)$ and $f_{pre-trained}(\cdot)$ stand for the current and the pre-trained deep neural networks, respectively. Moreover, N denotes the number of samples in the mini-batch.

Furthermore, similar to the N-pair loss function [30], the objective function described in Equation 3.5 is minimized by not only the direction of $D_{i,j}$ but also its magnitude. However, only the direction is important to calculate the distance between samples. Although normalization seems like a solution, enforcing being less than 1 on $D_{i,j}$ makes optimization difficult. Instead of that, L^2 regularization term of the N-pair embedding loss function is added on the proposed objective function:

$$J_{reg} = \frac{1}{N}\sum_{i=1}^{N}\|f_{\boldsymbol{\theta}}(\boldsymbol{x_i})\|_2 \tag{3.7}$$

Finally, the proposed loss function becomes as the following equation:

$$J_{centralized-mb-reg} = J_{\text{centralized-mb}} + \lambda J_{\text{reg-pre}} + \gamma J_{reg} \tag{3.8}$$

where λ and γ are regularization constants.

3.6 Experimental Results

In this section, the effectiveness of the proposed loss and batch construction method is demonstrated by the conducting experiments on three widely-used benchmark datasets for retrieval and clustering tasks. The proposed method has been shown to yield competitive or better results while comparing with the state-of-the-art approaches.

3.6.1 Benchmark Datasets

In this scope of the thesis, the results are obtained by utilizing three public benchmark datasets. Conventional protocol of splitting training and testing are followed for all datasets:

- **CUB-200-2011** [2] dataset consists of 200 species of birds with 11,788 images. Whereas the first 100 species (5,864 images) are split for training, the rest of 100 species (5,924 images) are used for testing.

- **Cars196** [3] dataset contains 196 classes of cars with 16,185 images. The first 98 classes (8,054 images) are used for training and remaining 98 classes (8,131 images) are reserved for testing.

- **Stanford Online Products** [1] dataset has 22,634 classes with 120,053 product images. The first 11,318 classes (59,551 images) are split for training and the other 11,316 (60,502 images) classes are used for testing.

3.6.2 Evaluation Metrics

Standard metric learning experimental protocol defined in [1] is followed to evaluate the proposed method for retrieval and clustering performance. Conventional k-means clustering [73] is used for the clustering task. In order to evaluate clustering performance, normalized mutual information (NMI) [45] and F_1 score is utilized. NMI could be described as follows where Ω and $\mathbb{C}$ denote a computed set of the clusters and a set of ground truth classes, respectively:

$$\mathrm{NMI}(\Omega, \mathbb{C}) = \frac{2I(\Omega; \mathbb{C})}{H(\Omega)H(\mathbb{C})} \tag{3.9}$$

where $H(\cdot)$ and $I(\cdot, \cdot)$ denote entropy and mutual information functions, respectively. NMI computes the label agreement between Ω and $\mathbb{C}$ neglecting the permutations whereas F_1 measures harmonic mean of precision, P, and recall, R, as follows:

$$\mathrm{F}_1 = \frac{2PR}{P + R} \tag{3.10}$$

Furthermore, Recall@K metric [74] is exploited for the retrieval task. At first, each image (query) in the test set retrieves K nearest neighbors from the test dataset. After that, Recall@K score gives 1, if there is at least one sample that has the same label as the query. Otherwise, it produces 0.

3.6.3 Training Setup

Tensorflow [62] deep learning library is used throughout the experiments. After images are normalized to 256x256, 224x224 random crop and data augmentation by horizontal mirroring are performed for standard pre-processing. GoogLeNet [25] which is pretrained on ImageNet ILSVRC dataset [75] are utilized as deep metric learning framework and the last layer (fully connected layer) is initialized by random weights. The weights of the randomly initialized layer are learned by utilizing 10 times larger learning rate than the parameters of the other layers for fast convergence. The base learning rate is selected as 10^{-4} for Stanford Online Products dataset whereas 10^{-5}

learning rate is utilized to train CUB-200-2011 and Cars196 since they tend to meet overfitting problem due to the limited dataset size. Adam [66] optimizer is used for mini-batch gradient descent.

The embedding size of the samples is fixed at 512 throughout the experiment since [1] shows that the embedding size does not have a key role on the performance of the deep metric learning framework. Moreover, regularization constants, λ and γ, described in Section 3.5 are empirically selected as 0.3 and 0.02, respectively. In addition to that, weight decay constant [76] and dropout [77] are empirically fixed to 0.01 and 0.2, respectively, in order to prevent overfitting and increase generalization. The maximum training iteration and mini-batch size are selected as 20,000 (10,000 with N-pair sampling and the rest by utilizing local neighborhood sampling) and 128, respectively, for a fair comparison with the existing state of the art deep learning methods. The key parameter K that determines how many k-nearest examples are sampled from each center is fixed to 16 regarding the results which are discussed in the following section.

3.6.4 Effect of the Parameter K

As mentioned in Section 3.4, increasing K value provides exploiting more representative positive and negative examples in the k-nearest neighborhood. On the other hand, it causes a mini-batch to become more correlated and it decreases the performance of the mini-batch gradient descent. Although a recent study [72] shows that utilizing batch renormalization reduces this performance degradation, it is not applied to be fair while comparing with the other metric learning methods since the most of them also generate correlated mini-batches to ensure sampling adequate number of hard negatives.

The motivation behind this trade-off is validated by the experimental results of the proposed loss function for variable K values on Stanford Online Product dataset which is indicated in Table 3.1. Whereas the best clustering and retrieval performances are obtained when $K = 16$, despite exploiting more representative positive and negative examples, the proposed method when $K = 128$ suffer from a bad local convergence. In the light of this results, K is selected as 16 for the rest of the experiments.

Table 3.1: Clustering and retrieval performances (%) of the proposed algorithm for different K values on the test set of Stanford Online Products dataset[1] @20k iterations. The best results are indicated in bold.

K	NMI	F_1	R@1	R@10	R@100
4	90.7	39.3	72.6	86.5	94.4
8	90.7	39.5	72.7	86.7	94.4
16	**90.8**	**40.3**	**73.5**	**87.3**	**94.5**
32	90.5	38.8	72.6	86.6	94.1
64	90.4	39.0	72.8	86.8	94.3
128	90.1	36.9	70.2	84.8	93.1

3.6.5 Ablation Studies

In this section, the importance of the proposed sampling method and the effect of regularization term on the difference between produced embedding vectors and pre-computed embedding vectors are highlighted.

Table 3.2: Clustering and retrieval performances (%) of the proposed loss function with different sampling approaches on the test set of Stanford Online Products dataset[1]. The best results are indicated in bold.

Method	NMI	F_1	R@1	R@10	R@100
N-pair [30]	87.9	27.1	66.4	82.9	92.1
Proposed Loss + $P = 4$	87.4	25.3	63.4	81.2	92.2
Proposed Loss + Sampling	**90.8**	**40.3**	**73.5**	**87.3**	**94.5**

At first, the proposed loss function is trained on the train set of Stanford Online Products dataset by selecting P positive examples of each class in a batch without utilizing local neighborhood sampling. It is important to recall that when P is equal to 2, the loss function is the same with N-pair loss function. For different P values which are 2 and 4, clustering and retrieval performances of the proposed loss function are given in Table 3.2. These results demonstrate the importance of training with as many negative classes as possible by using N-pair sampling method since performances are reduced while increasing P. However, the local neighborhood sampling provides to select representative positive and negative examples and it increases clustering and retrieval performances.

Furthermore, the proposed loss function is trained on the train set of Stanford Online Products dataset without a regularization term which is indicated in Equation 3.6. Then, two different evaluation processes are followed. Firstly, the embedding vectors are produced and evaluated in a conventional way. Secondly, while fixing k-nearest neighborhood of each example according to $\boldsymbol{Z}_{pre-trained}$, only retrieval performance of the network is computed. As seen in Table 3.3, while retrieval task for each example is restricted to samples which are selected from precomputed k-nearest neighborhood, R@1 metric increases by up to 2.4 %. It validates that the embedding representations of negative samples are mapped into undesired far points from the center due to calculating a neighborhood information in $\boldsymbol{Z}_{pre-trained}$ only once to reduce computational complexity instead of each iteration passing all training samples through the network. In addition to that, Table 3.3 demonstrates that regularizing L^2 norm of the difference between produced embedding vectors and precomputed embedding vectors eliminates this deficiency.

Table 3.3: Clustering and retrieval performances (%) of the proposed loss function with different sampling approaches on the test set of Stanford Online Products dataset[1]. The best results are indicated in bold.

Method	NMI	F_1	R@1	R@10	R@100
Proposed Loss + Sampling	90.0	37.5	71.4	86.2	94.0
Proposed Loss + Sampling by fixing k-nearest neighborhood	-	-	**73.8**	86.5	-
Proposed Loss + Sampling + Regularization	**90.8**	**40.3**	73.5	**87.3**	**94.5**

3.6.6 Baseline Methods

The deep metric learning framework described in [1] is applied on three aforementioned public datasets for a direct comparison with state-of-the art methods in the literature that include contrastive embedding [28], triplet embedding [23] with N-pair sampling and the more recent lifted structure [1], N-pair loss [30], clustering via facility location [15] and angular loss [31].

Furthermore, recently, Duan et al. [32] proposed novel deep metric learning framework based on an *adversarial* learning [78]. In [32], a deep adversarial metric learning framework (DAML) generates synthetic hard negatives from the observed negative samples to boost the performance of the existing metric learning loss functions. Although it could be easily adapted for the proposed loss function in this section, it exceeds the scope of this thesis. Nevertheless, in order to preserve the integrity of the metric learning literature, DAML with N-pair loss function that has current state-of-the-art results is also selected as one of the baseline methods. Besides, note that DAML is trained through 40,000 iterations (20,000 for pre-training with N-pair loss) instead of 20,000 iterations like proposed and the other baseline methods.

3.6.7 Quantitative Results

Tables 3.4, 3.5 and 3.6 indicate the quantitative results of the proposed method compared with baseline methods on CUB-200-2011, Cars196 and Stanford Online Product datasets, respectively, for clustering and retrieval tasks. The red values are utilized to indicate best performances whereas the second best results are indicated in blue. F_1 performance of clustering via facility location approach is not given since it is not available in its original paper [15].

At first, it is required to compare the proposed method with N-pair loss function since the introduced loss function is equivalent to N-pair objective function when the number of positive samples for each anchor in the mini-batch is 1. Although N-pair embedding yields impressive results, the proposed loss function combined with the introduced neighborhood sampling outperforms N-pair on all three datasets in terms of clustering and retrieval performance by up to 3.9 %, 13.2% and 7.4% points on

NMI, F_1 and R@1 metrics, respectively.

It demonstrates that the proposed sampling method is efficient to select representative hard negative and positive samples and it leads the proposed loss function to exploit more than one positive example while keeping dissimilar samples away unlike N-pair. It is significant to notice that the large gap between N-pair and proposed method in terms of F_1 evaluation metric on Stanford Online Products dataset indicates that proposed method considers not only recall but also precision performance by getting all positive examples closer to each other while pushing negative samples away in the k-nearest neighborhood.

Furthermore, the proposed method yields state-of-the-art results in terms of NMI, F_1 and R@K scores on CUB-200-2011, Cars196 and Stanford Online Products datasets. The largest performance margins by 1.4 % on NMI metric, 7.9 % on F_1 score and 5.1 % R@1 metric between the proposed approach and current state-of-the-art method, DAML, are obtained on Stanford Online Products dataset that is the most challenging one of the utilized datasets as each cluster has approximately 5.3 images. It is expected since local neighborhood sampling approach could extract adequate number of hard negatives even if K is selected as a small value.

Table 3.4: Clustering and retrieval performances (%) of the proposed algorithm on the test set of CUB-200-2011 dataset[2]. The best results are indicated in red whereas the second best results are showed in blue. (Best viewed in color.)

Method	NMI	F_1	R@1	R@2	R@4	R@8
Contrastive [28]	47.2	12.5	27.2	36.3	49.8	62.1
Triplet [23] + N-pair sampling	54.1	20.0	42.8	54.9	66.2	77.6
Lifted [1]	56.4	22.6	46.9	59.8	71.2	81.5
N-pair [30]	60.2	28.2	51.9	64.3	74.9	83.2
Facility [15]	59.2	-	48.2	61.4	71.8	81.9
Angular [31]	61.0	**30.2**	**53.6**	65.0	75.3	83.7
DAML with N-pair loss [32]	**61.3**	29.5	52.7	**65.4**	**75.5**	**84.3**
Proposed Method	**63.3**	**31.1**	**55.4**	**66.8**	**77.3**	**85.8**

Table 3.5: Clustering and retrieval performances (%) of the proposed algorithm on the test set of Cars196 dataset[3]. The best results are indicated in red whereas the second best results are showed in blue. (Best viewed in color.)

Method	NMI	F_1	R@1	R@2	R@4	R@8
Contrastive [28]	42.3	10.5	27.6	38.3	51.0	63.9
Triplet [23] + N-pair sampling	54.3	19.6	46.3	59.9	71.4	81.3
Lifted [1]	57.8	25.1	59.9	70.4	79.6	87.0
N-pair [30]	62.7	31.8	68.9	78.9	85.8	90.9
Facility [15]	59.0	-	58.1	70.6	80.3	87.8
Angular [31]	62.4	31.8	71.3	80.7	87.0	91.8
DAML with N-pair loss [32]	**66.0**	**36.4**	**75.1**	**83.8**	**89.7**	**93.5**
Proposed Method	**66.6**	**36.7**	**76.3**	**85.1**	**90.8**	**94.6**

Table 3.6: Clustering and retrieval performances (%) of the proposed algorithm on the test set of Stanford Online Products dataset[1]. The best results are indicated in red whereas the second best results are showed in blue. (Best viewed in color.)

Method	NMI	F_1	R@1	R@10	R@100
Contrastive [28]	82.4	10.1	37.5	53.9	71.0
Triplet [23] + N-pair sampling	86.3	20.2	53.9	72.1	85.7
Lifted [1]	87.2	25.3	62.6	80.9	91.2
N-pair [30]	87.9	27.1	66.4	82.9	92.1
Facility [15]	**89.5**	-	67.0	**83.7**	**93.2**
Angular [31]	87.8	26.5	67.9	83.2	92.2
DAML with N-pair loss [32]	89.4	**32.4**	**68.4**	83.5	92.3
Proposed Method	**90.8**	**40.3**	**73.5**	**87.3**	**94.5**

3.6.8 Qualitative Results

Figures 3.4, 3.5 and 3.6 visualize the t-SNE [8] 2-D plots on the embedding vectors which are produced by proposed method on CUB-200-2011 [2], Cars196 [3] and Stanford Online Products [1] datasets, respectively. Various representative clusters are highlighted by magnifying corresponding regions. The best view could be obtained on a monitor when zoomed in. As seen in figures, in spite of high variations in pose, appearance and view point, the proposed approach produces mapping into a compact embedding space that maintains semantic similarity.

Figures 3.7 and 3.8 compare the proposed loss function with N-pair in terms of top-7 retrieval performance on Stanford Online Products dataset. The green and red borders surrounding retrieved images indicate whether they belong to the same class as the queried image or not, respectively. As seen in Figure 3.7, most failures of the proposed method compared to N-pair stems from falsely retrieved images that have almost identical classes of the queried images. For example, at the one and second rows of the Figure 3.7, whereas N-pair retrieves four correct purple fans, the proposed method retrieves only one true image but the other retrieved images are also purple fans.

On the other hand, the efficiency of the proposed approach on identifying similar examples is demonstrated in Figure 3.8. Since N-pair considers only one positive sample while keeping negative examples away, it tends to fail in retrieving more than one similar samples. For instance, at the third and fourth rows of the Figure 3.8, whereas the second similar example of the queried image is the fourth closest sample in the embedding space which is produced by N-pair, the top-6 images retrieved by the proposed method contain six successful matches.

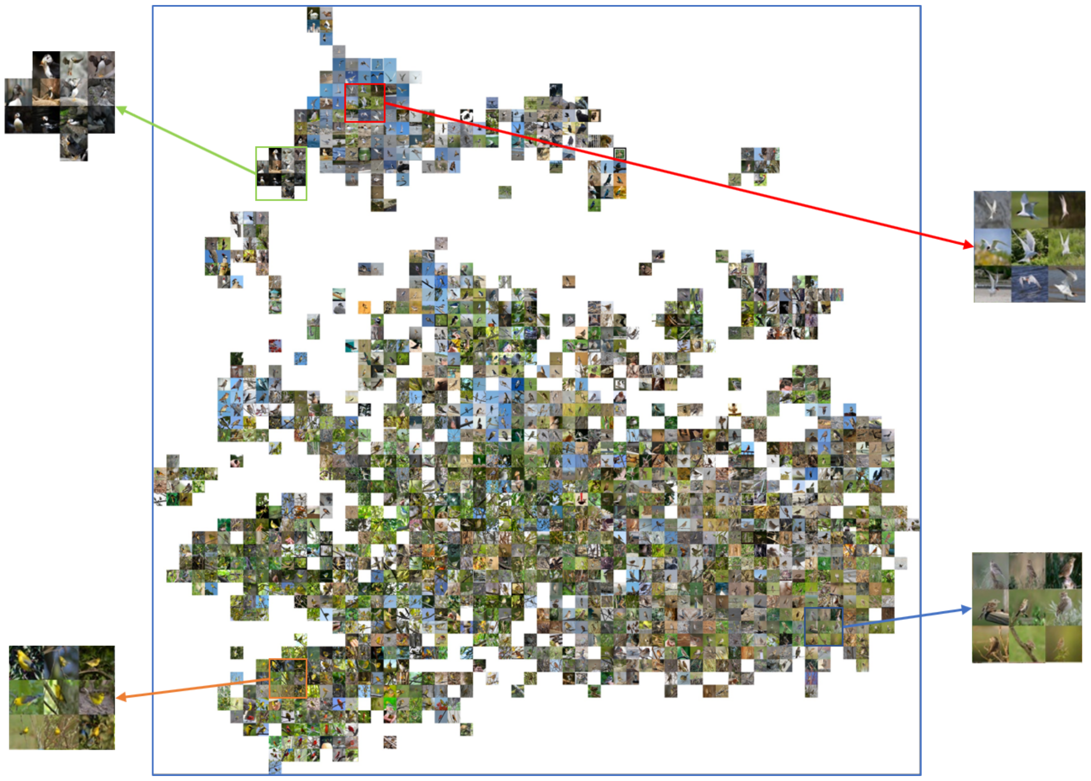


Figure 3.4: Barnes Hut t-SNE visualization [8] of the proposed embedding method on the test set of CUB-200-2011 dataset. (Best viewed on a monitor when zoomed in.)

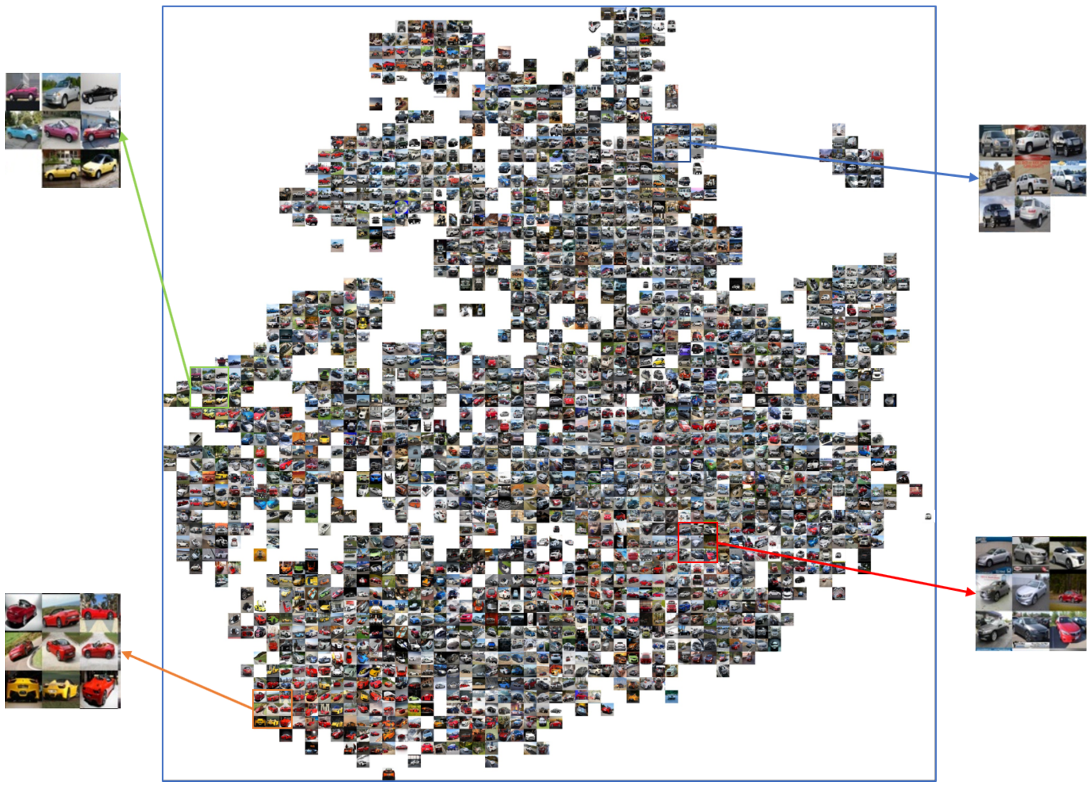


Figure 3.5: Barnes Hut t-SNE visualization [8] of the proposed embedding method on the test set of Cars196 dataset. (Best viewed on a monitor when zoomed in.)

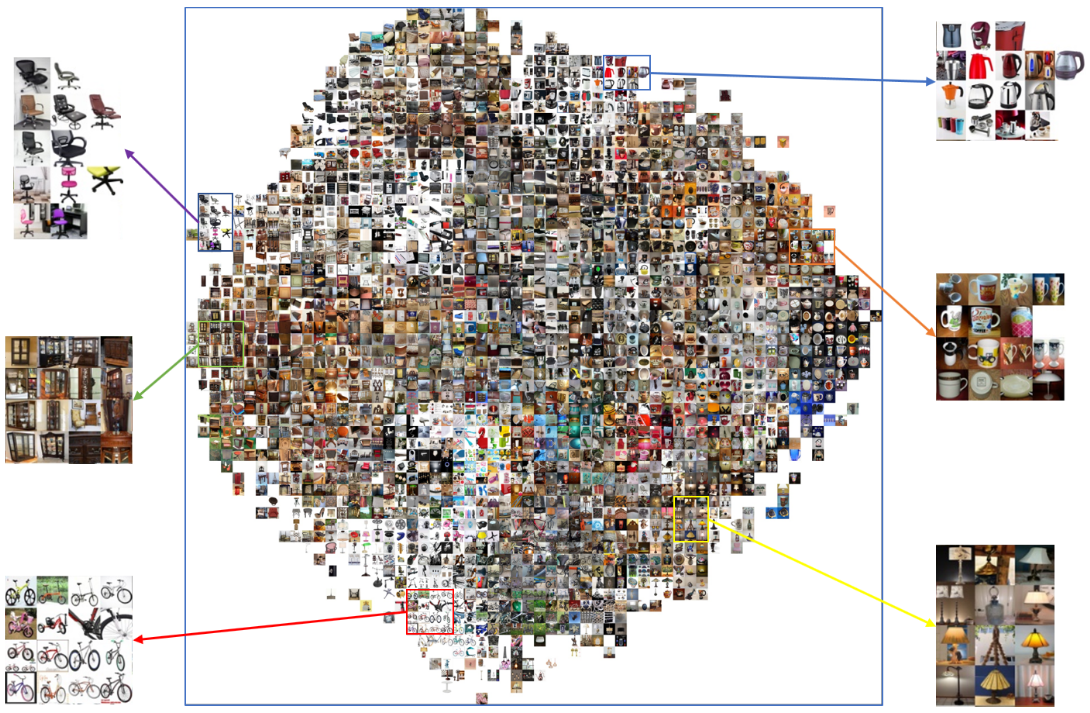


Figure 3.6: Barnes Hut t-SNE visualization [8] of the proposed embedding method on the test set of Stanford Online Products dataset. (Best viewed on a monitor when zoomed in.)

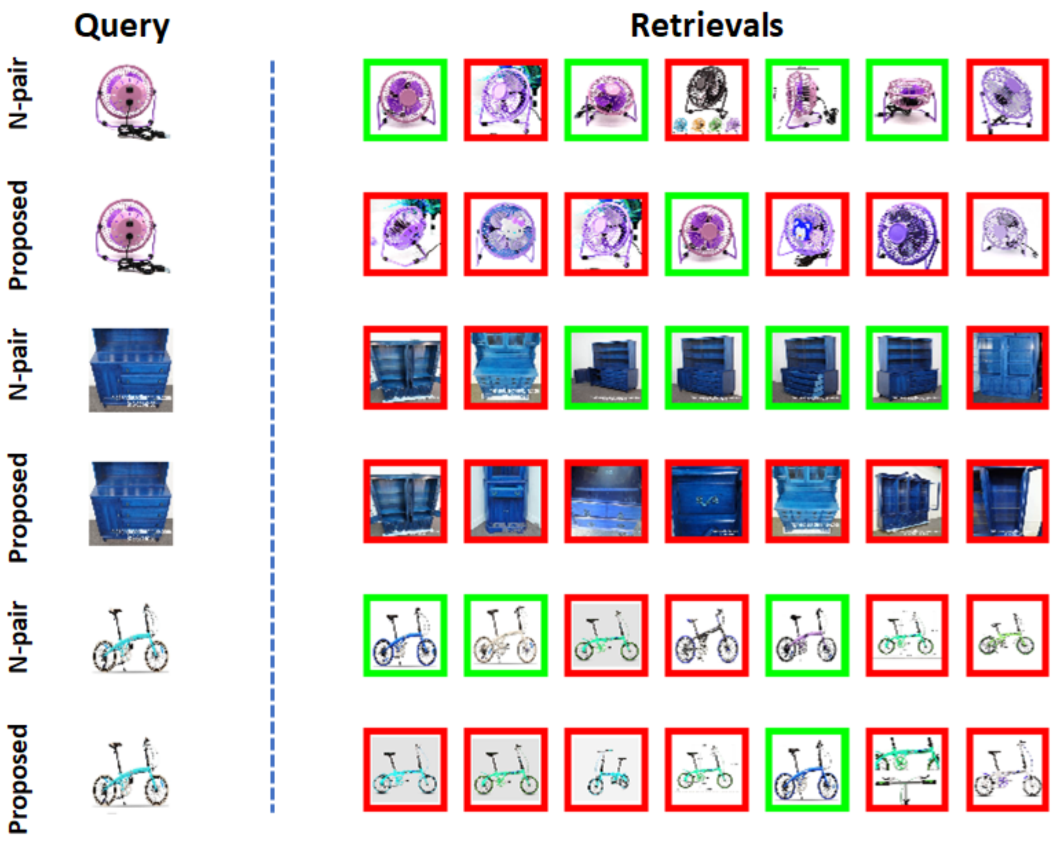


Figure 3.7: Visualization of top-7 retrieval results from queried images where N-pair performs better than the proposed method on the test set of Stanford Online Products dataset. The retrieved images that belongs to the same class with the queried image is indicated in green border whereas retrieved images which are dissimilar to the queried image is shown in red borders. (Best viewed in color.)

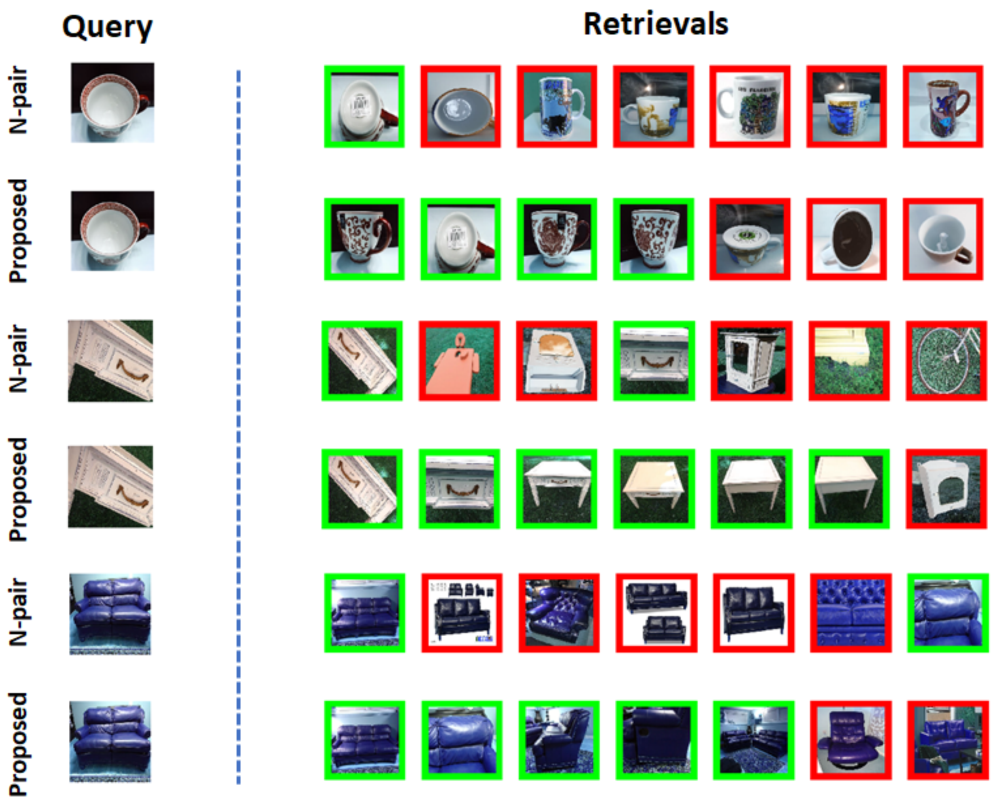


Figure 3.8: Visualization of top-7 retrieval results from queried images where proposed method performs better than N-pair on the test set of Stanford Online Products dataset. The retrieved images that belongs to the same class with the queried image is indicated in green border whereas retrieved images which are dissimilar to the queried image is shown in red borders. (Best viewed in color.)

CHAPTER 4

DEEP METRIC LEARNING FOR RECURRENT NEURAL NETWORKS

In this chapter of the thesis, a novel deep metric learning framework with RNN is presented by utilizing loss function and sampling method which are introduced in Chapter 3. Following the explanation of the motivation behind this chapter, the novel deep metric learning framework with RNN is introduced. Finally, the efficiency of the proposed framework is validated with quantitative and qualitative experimental results.

4.1 Motivation

In Chapter 3, the novel loss function and the batch construction method is proposed. Existing approaches in the literature [1, 9, 15, 28, 29, 30, 31, 32] and also proposed method does not consider high-order correlations in the dataset. Except angular embedding loss explained in Section 2.1.2.5, deep metric learning models are optimized with respect to distance-based loss functions that carries second-order information [31] whereas angular embedding loss utilizes angle information which defines the third-order triangulation among three points. Although angular objective function exploits third-order relation between samples, the solution space may be higher order.

RNN described in Section 2.2.6 is a powerful deep learning tool to model long-term dependencies from the sequential data. However, this powerful tool could be also utilized for sequential processing in absence of sequences [79, 80]. In [80], RNN recognizes house numbers by reading images from left to right. In addition to that, [80] uses RNN to produce images which contain digits by sequentially adding color to a canvas.

The local neighborhood sampling method which is introduced in Section 3.4 generates mini-batches which consist of hard negatives and representative positives of the selected centers. In other words, it creates a k-nearest neighborhood manifold for each center so the high-order correlation between samples in the same manifold could be modeled by using RNN. In order to input center and samples in its k-nearest neighborhood in a sequential manner, the simple trick is applied. Each sequence is generated by sorting the examples in a descending order regarding the distance to its center.

4.2 Proposed Deep Metric Learning Framework with RNN

While training the deep metric learning framework described in Section 3.2 by utilizing proposed loss function or one of existing deep metric learning objective functions in the literature, the high-order relations between samples are ignored [31] since they optimize the network with respect to distances (second-order correlation) or angles (third-order relation) between samples.

Recall the proposed $J_{k\text{-nearest-mb}}$ objective function that is trained by utilizing local neighborhood sampling introduced in Section 3.4 where $D_{i,j} = f_{\boldsymbol{\theta}_{cnn}}(\boldsymbol{x_i})^T f_{\boldsymbol{\theta}_{cnn}}(\boldsymbol{x_j})$:

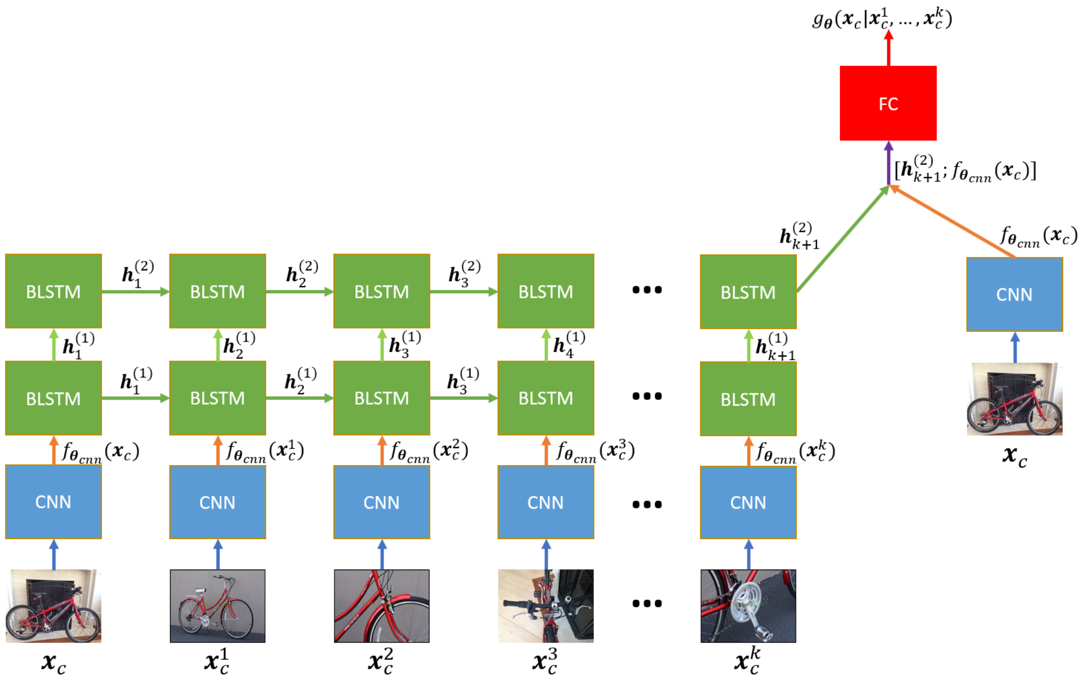
$$J_{k\text{-nearest-mb}} = \frac{1}{|\mathbb{C}|} \sum_{c \in \mathbb{C}} - \log \frac{\frac{1}{|\mathbb{P}_k|} \sum_{i \in \mathbb{P}_k} \exp\{D_{c,i}\}}{\sum_{k \in \mathbb{K}_c} \exp\{D_{c,k}\}} \tag{4.1}$$

where $\mathbb{C}$ denotes the set of centers in the mini-batch and $\mathbb{K}_c$ is constructed from K closest examples to each center at index c in the embedding space that is mapped by $f_{pre-trained}(\cdot)$. Besides, $\mathbb{P}_k$ indicates a set of positive examples of the center, c, in $\mathbb{K}_c$.

As seen in Equation 4.1, the loss function is optimized by utilizing distances between the center and examples that belong to its k-nearest neighborhood which create a manifold. It provides that determining which samples are required to model high-order relations among them for mapping hard negatives in far away points from the center since the other negative samples could be mapped into the desired points in the embedding space by utilizing conventional deep metric learning framework.

By inspiring from encoder-decoder models described in Section 2.2.7, bidirectional LSTM encoder which is used to capture high-order correlations between samples in the constructed manifold. The overview of the proposed deep metric learning framework by utilizing bidirectional LSTM encoder is illustrated in Figure 4.1. Each input sequence of bidirectional LSTM could be generated as sorting the embedding representations of the examples which are produced by applying $f_{\boldsymbol{\theta}_{cnn}}(\cdot)$ in a descending order regarding the distance to its center:

$$\boldsymbol{X}_c^i = \{f_{\boldsymbol{\theta}_{cnn}}(\boldsymbol{x}_c), f_{\boldsymbol{\theta}_{cnn}}(\boldsymbol{x}_c^1), f_{\boldsymbol{\theta}_{cnn}}(\boldsymbol{x}_c^2), ..., f_{\boldsymbol{\theta}_{cnn}}(\boldsymbol{x}_c^k)\} \tag{4.2}$$



$g_{\boldsymbol{\theta}}(\boldsymbol{x}_c|\boldsymbol{x}_c^1, ..., \boldsymbol{x}_c^k)$
FC
$[\boldsymbol{h}_{k+1}^{(2)}; f_{\boldsymbol{\theta}_{cnn}}(\boldsymbol{x}_c)]$
$\boldsymbol{h}_{k+1}^{(2)}$
$f_{\boldsymbol{\theta}_{cnn}}(\boldsymbol{x}_c)$
BLSTM
$\boldsymbol{h}_1^{(2)}$
$\boldsymbol{h}_2^{(2)}$
$\boldsymbol{h}_3^{(2)}$
$\boldsymbol{h}_1^{(1)}$
$\boldsymbol{h}_2^{(1)}$
$\boldsymbol{h}_3^{(1)}$
$\boldsymbol{h}_4^{(1)}$
$\boldsymbol{h}_{k+1}^{(1)}$
$f_{\boldsymbol{\theta}_{cnn}}(\boldsymbol{x}_c)$
$f_{\boldsymbol{\theta}_{cnn}}(\boldsymbol{x}_c^1)$
$f_{\boldsymbol{\theta}_{cnn}}(\boldsymbol{x}_c^2)$
$f_{\boldsymbol{\theta}_{cnn}}(\boldsymbol{x}_c^3)$
$f_{\boldsymbol{\theta}_{cnn}}(\boldsymbol{x}_c^k)$
CNN
$\boldsymbol{x}_c$
$\boldsymbol{x}_c^1$
$\boldsymbol{x}_c^2$
$\boldsymbol{x}_c^3$
$\boldsymbol{x}_c^k$

Figure 4.1: Proposed Deep Metric Learning Framework that utilizes both of CNN and RNN. (Best viewed in color.)

Therefore, the context of each input sequence could be mapped into a fixed-length vector which is the final hidden layer of bidirectional LSTM. This is common technique that is used in LSTM encoder-decoder architectures to encode the long-term dependencies [56, 57, 55]. The hidden units in the hidden layer i of the proposed architecture, $\boldsymbol{h}_t^{(i)}$, could be calculated by utilizing the bidirectional LSTM formulation that is detailedly explained in Section 2.2.6 where $\boldsymbol{x}_t \in \boldsymbol{X}_c^i$.

After calculating the final hidden layer of the bidirectional LSTM, $\boldsymbol{h}_{k+1}^{(2)}$, where the number of hidden layers is 2 in this architecture, the context information of the input sequence could be passed through the network by concatenating $\boldsymbol{h}_{k+1}^{(2)}$ and the embedding representation of the center example that is produced by CNN, $f_{\boldsymbol{\theta}_{cnn}}(\boldsymbol{x}_c)$, along the last dimension. The concatenation operator, $[.;.]$, is highly used in the networks with attention mechanism which is described in Section 2.2.7 to pass attention information through the decoder.

Finally, the embedding representations of each sample could be produced by applying the proposed architecture, $g_{\boldsymbol{\theta}}(.|.)$, that contains RNN and CNN models where $\boldsymbol{\theta}$ denotes the parameters of the architecture:

$$g_{\boldsymbol{\theta}}(\boldsymbol{x}_c|\boldsymbol{x}_c^1, ..., \boldsymbol{x}_c^k) = \boldsymbol{W}^{(e)}[\boldsymbol{h}_{k+1}^{(2)}; f_{\boldsymbol{\theta}_{cnn}}(\boldsymbol{x}_c)] + \boldsymbol{b}^{(e)} \tag{4.3}$$

In the training phase of the proposed deep metric learning model with RNN, the same procedure with the proposed method based on CNN described in the Chapter 3 is followed by exploiting Equation 3.8 where the distances between samples are calculated using the condition on their k-closest examples:

$$D_{i,j} = g_{\boldsymbol{\theta}}(\boldsymbol{x}_i|\boldsymbol{x}_i^1, ..., \boldsymbol{x}_i^k)^T g_{\boldsymbol{\theta}}(\boldsymbol{x}_j|\boldsymbol{x}_j^1, ..., \boldsymbol{x}_j^k) \tag{4.4}$$

On the other hand, in the inference phase, while the proposed framework produces the embedding vector of each sample, its k-nearest samples are required in addition to the queried example. This k-closest information could easily be extracted from stored embeddings, $\boldsymbol{Z}_{pre-trained}$, which are generated by $f_{pre-trained}(\cdot)$.

4.3 Experimental Results

Following the experimental protocol described in Section 3.6, the novel proposed metric learning framework with RNN is evaluated on CUB-200-2011 [2], Cars196 [3] and Stanford Online Products [1] datasets for clustering and retrieval tasks. The training and testing split procedures are mentioned in Section 3.6.1. NMI, F_1 and R@K scores where are detailedly explained in Section 3.6.2 are utilized to evaluate clustering and retrieval performance.

Tensorflow [62] deep learning library is used throughout the experiments. The number of hidden layers in RNN architecture is empirically selected as 2. The number of hidden units in each bidirectional LSTM is fixed to 512 for a consistency with the embedding size. Whereas the half number of hidden units is reserved for left-to-right LSTM, the rest is exploited as right-to-left LSTM. The dropout [77] between LSTM layers is fixed to 0.5 as default. The parameters of CNN architecture is kept same with Section 3.6.3 for a fair comparison. The new layer which is added to produce embedding vectors which have a predetermined size is randomly initialized.

The maximum training iteration and mini-batch size are selected as 40,000 (20,000 with proposed and sampling method by using conventional CNN architecture whose results are indicated in Section 3.6) and 128, respectively. K is fixed to 16 considering observations in Section 3.6.4 during both of training and inference processes.

For a fair comparison with the conventional deep metric learning framework, it is also trained by using the proposed method through 40,000 iterations while the pre-computed embedding vectors are produced by the network trained for 20,000 iterations. The results on the test set of Stanford Online Products dataset are indicated in Table 4.1. The clustering and retrieval performances are almost fixed during network training. It shows that the conventional framework has been already converged in 20,000 iterations. Hence, the proposed method which is trained by utilizing deep metric learning framework based on only CNN @20k iterations is picked to compare with the proposed RNN based framework.

Furthermore, the introduced framework is also compared with baseline methods described in Section 3.6.6 in terms of clustering and retrieval performances.

Table 4.1: Clustering and retrieval performances (%) of the proposed algorithm described in Chapter 3 for the CNN network trained through different iterations on the test set of Stanford Online Products dataset[1]. The best results are indicated in bold.

Iteration	NMI	F_1	R@1	R@10	R@100
20000	**90.8**	**40.3**	**73.5**	**87.3**	**94.5**
25000	90.6	39.9	73.3	86.7	94.1
30000	90.7	40.1	73.4	87.0	94.2
35000	90.3	40.2	72.3	86.4	94.1
40000	90.5	40.1	73.0	86.8	94.2

4.3.1 Ablation Study

In this section, the importance of sorting examples in a descending order regarding the distance to its center is highlighted. Table 4.2 validates the efficiency of ordering inputs. While a sorting trick is not used, clustering and retrieval performances of the proposed algorithm by utilizing RNN framework on the test set of Stanford Online Products dataset decreases from even initial performances which are produced by only exploiting CNN.

Table 4.2: Clustering and retrieval performances (%) of the proposed algorithm by utilizing RNN framework with and without ordering inputs on the test set of Stanford Online Products dataset[1]. The best results are indicated in bold.

Method	NMI	F_1	R@1	R@10	R@100
Proposed Method	90.8	40.3	73.5	87.3	94.5
Proposed Method + RNN without ordering inputs	90.2	37.4	70.2	85.2	93.5
Proposed Method + RNN	**91.4**	**42.8**	**74.3**	**87.9**	**95.2**

4.3.2 Quantitative Results

Tables 4.3, 4.4 and 4.5 indicate the quantitative results of the proposed framework compared with the conventional framework which is trained by using baseline methods on CUB-200-2011 [2], Cars196 [3] and Stanford Online Products [1] datasets for clustering and retrieval tasks, respectively. The red values are utilized to indicate best performances whereas the second best results are indicated in blue. F_1 performance of clustering via facility location approach is not given since it is not available in its original paper [15].

Since the superior performance of the proposed method over the existing state-of-the-art methods by using the conventional deep metric learning framework is thoroughly discussed in Section 3.6.7, it is important to notice the clustering and retrieval performances of the conventional and the proposed frameworks.

As seen in Tables 4.3, 4.4 and 4.5, the proposed framework outperforms the conventional model with margin by up to 0.8% on NMI metric, 2.5 % on F_1 score and 1.0 % R@1 metric. By combining these results with observations in Table 4.1, it could be easily deduced that utilizing RNN boosts the performance of the retrieval and clustering tasks since it could model higher-order relations in the k-nearest neighborhood while the conventional framework could not continue learning.

Table 4.3: Clustering and retrieval performances (%) of the proposed algorithm with the RNN framework on the test set of CUB-200-2011 dataset[2]. The best results are indicated in red whereas the second best results are showed in blue. (Best viewed in color.)

Method	NMI	F_1	R@1	R@2	R@4	R@8
Contrastive [28]	47.2	12.5	27.2	36.3	49.8	62.1
Triplet [23] + N-pair sampling	54.1	20.0	42.8	54.9	66.2	77.6
Lifted [1]	56.4	22.6	46.9	59.8	71.2	81.5
N-pair [30]	60.2	28.2	51.9	64.3	74.9	83.2
Facility [15]	59.2	-	48.2	61.4	71.8	81.9
Angular [31]	61.0	30.2	53.6	65.0	75.3	83.7
DAML with N-pair loss [32]	61.3	29.5	52.7	65.4	75.5	84.3
Proposed Method	**63.3**	**31.1**	**55.4**	**66.8**	**77.3**	**85.8**
Proposed Method + RNN	**64.0**	**32.2**	**55.6**	**67.2**	**77.0**	**85.0**

Table 4.4: Clustering and retrieval performances (%) of the proposed algorithm with the RNN framework on the test set of Cars196 dataset[3]. The best results are indicated in red whereas the second best results are showed in blue. (Best viewed in color.)

Method	NMI	F_1	R@1	R@2	R@4	R@8
Contrastive [28]	42.3	10.5	27.6	38.3	51.0	63.9
Triplet [23] + N-pair sampling	54.3	19.6	46.3	59.9	71.4	81.3
Lifted [1]	57.8	25.1	59.9	70.4	79.6	87.0
N-pair [30]	62.7	31.8	68.9	78.9	85.8	90.9
Facility [15]	59.0	-	58.1	70.6	80.3	87.8
Angular [31]	62.4	31.8	71.3	80.7	87.0	91.8
DAML with N-pair loss [32]	66.0	36.4	75.1	83.8	89.7	93.5
Proposed Method	**66.6**	**36.7**	**76.3**	**85.1**	**90.8**	**94.6**
Proposed Method + RNN	**68.2**	**37.2**	**77.3**	**85.0**	**89.8**	**93.7**

Table 4.5: Clustering and retrieval performances (%) of the proposed algorithm with the RNN framework on the test set of Stanford Online Products dataset[1]. The best results are indicated in red whereas the second best results are showed in blue. (Best viewed in color.)

Method	NMI	F_1	R@1	R@10	R@100
Contrastive [28]	82.4	10.1	37.5	53.9	71.0
Triplet [23] + N-pair sampling	86.3	20.2	53.9	72.1	85.7
Lifted [1]	87.2	25.3	62.6	80.9	91.2
N-pair [30]	87.9	27.1	66.4	82.9	92.1
Facility [15]	89.5	-	67.0	83.7	93.2
Angular [31]	87.8	26.5	67.9	83.2	92.2
DAML with N-pair loss [32]	89.4	32.4	68.4	83.5	92.3
Proposed Loss + Sampling	**90.8**	**40.3**	**73.5**	**87.3**	**94.5**
Proposed Method + RNN	**91.4**	**42.8**	**74.3**	**87.9**	**95.2**

4.3.3 Qualitative Results

Figure 4.2 visualizes the t-SNE [8] 2-D plots on the embedding vectors which are produced by proposed method on Stanford Online Products [1] dataset. Various representative clusters are highlighted by magnifying corresponding regions. The best view could be obtained on a monitor when zoomed in. As seen in the figure, in spite of high variations in pose, appearance and view point, the proposed approach produces mapping into a compact embedding space that maintains semantic similarity.

Figures 4.4 and 4.3 compare the proposed framework that contains both of CNN and RNN architectures with the conventional framework that utilizes only CNN in terms of top-7 retrieval performance on Stanford Online Products dataset. The green and red borders surrounding retrieved images indicate whether they belong to the same class as the queried image or not, respectively.

As seen in Figure 4.3, most failures of the proposed framework compared to conventional one from falsely retrieved images that have similar context with the queried images. For example, at the fifth and sixth rows of the Figure 4.3, whereas conventional framework by using the proposed approach defined in Chapter 3 retrieves two correct fans which have different colors with the queried black and white fan, the proposed framework retrieves other black and white fans which belong to different class of the queried image.

On the other hand, the efficacy of the proposed framework on learning higher-order relations is demonstrated in Figure 4.4. Since the conventional framework considers only distance based second-order relations in data, it tends to fail in large pose and scale variations. However, introduced framework which contains RNN and CNN architectures is more robust to rotation and scaling transformations of the images owing to modeling long-term dependencies.

For instance, at the fifth and sixth rows of the Figure 4.4, whereas the conventional framework lost context information when large pose and scaling variations occur, the top-7 images retrieved by the proposed framework contain seven successful matches while implicitly walking on a learned manifold.

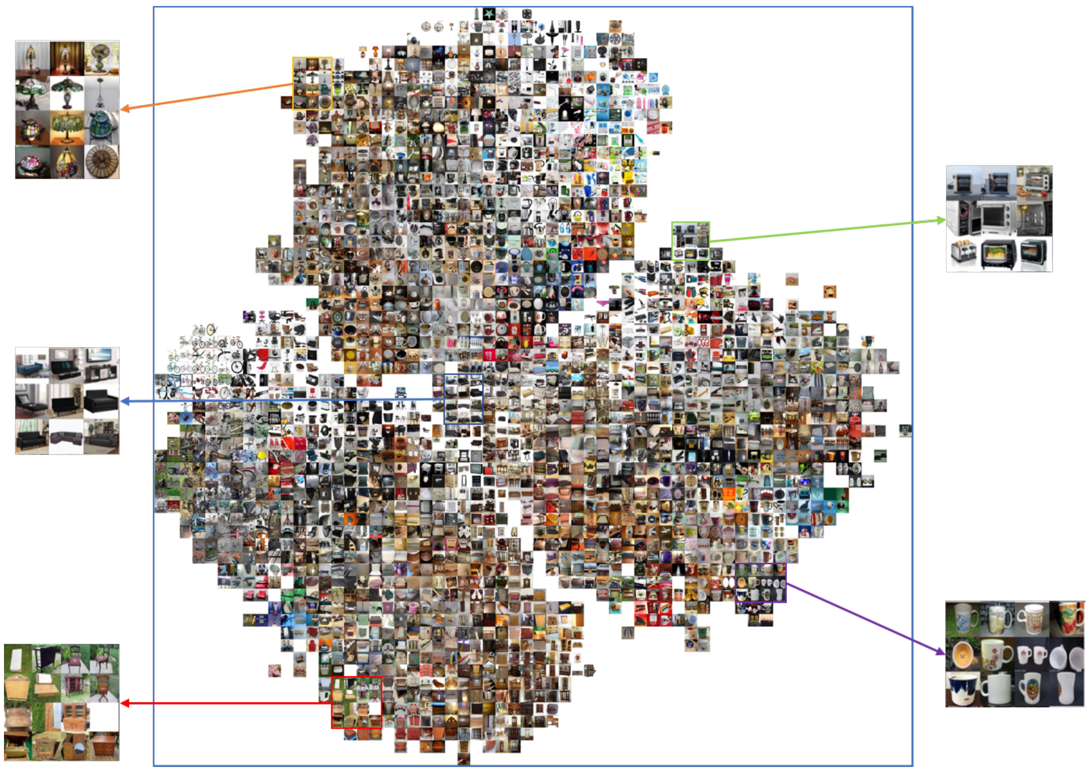


Figure 4.2: Barnes Hut t-SNE visualization [8] of the proposed embedding method with the RNN framework on the test set of Stanford Online Products dataset. (Best viewed on a monitor when zoomed in.)

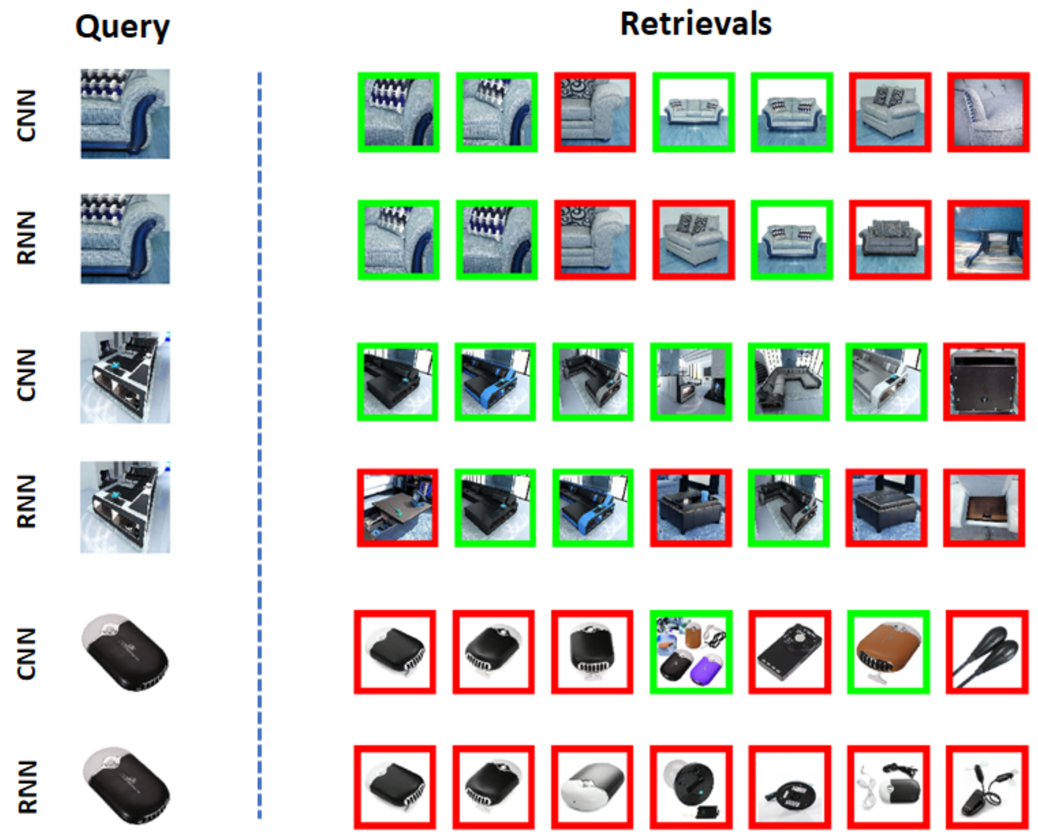


Figure 4.3: Visualization of top-7 retrieval results from queried images where the conventional framework based on CNN performs better than the proposed framework that contains RNN and CNN on the test set of Stanford Online Products dataset. The retrieved images that belongs to the same class with the queried image is indicated in green border whereas retrieved images which are dissimilar to the queried image is shown in red borders. (Best viewed in color.)

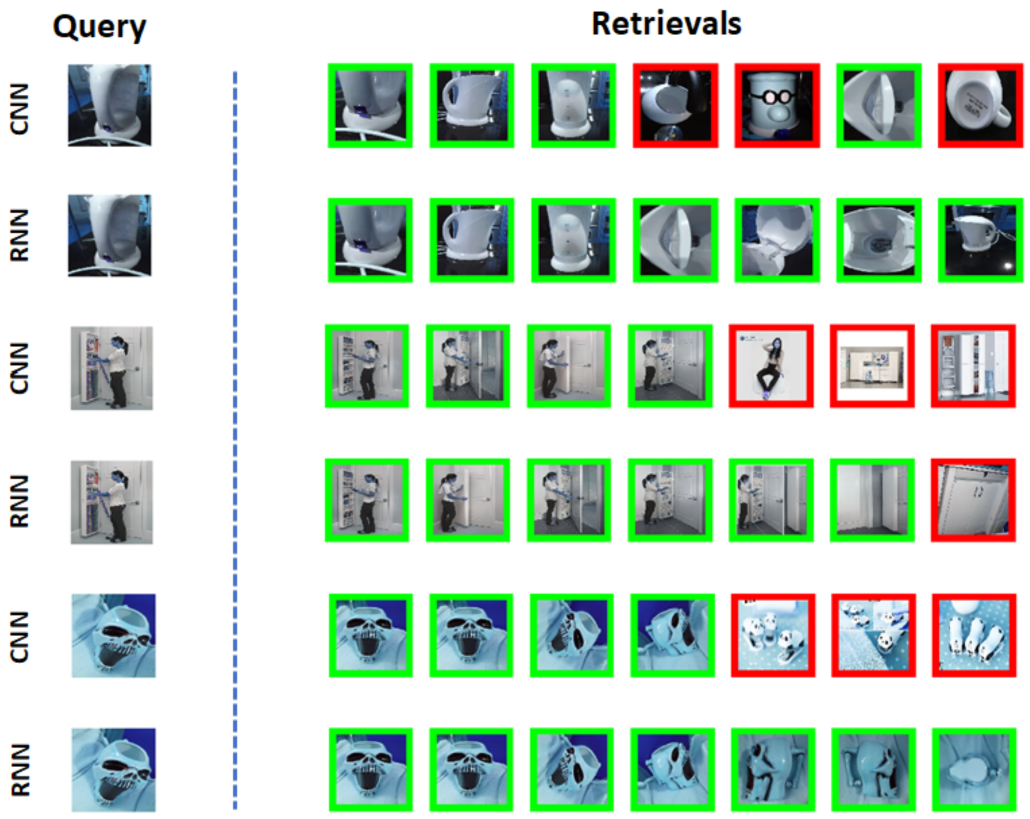


Figure 4.4: Visualization of top-7 retrieval results from queried images where the proposed framework that contains RNN and CNN performs better than the conventional framework based on CNN on the test set of Stanford Online Products dataset. The retrieved images that belongs to the same class with the queried image is indicated in green border whereas retrieved images which are dissimilar to the queried image is shown in red borders. (Best viewed in color.)

CHAPTER 5

CONCLUSIONS

Throughout this thesis study, a novel metric learning approach is proposed in order to address two main limitations of deep metric learning methods. Firstly, most of the existing approaches send negative samples away from the selected anchor by identifying only a pair of samples with the same label due to the inefficient mini-batch construction methods to pick representative hard negatives. In order to remove this deficiency, in Section 3.3, a novel loss function which is the extension of N-pair [30] embedding is presented. The proposed loss function identifies multiple similar examples from several negative samples. Following that, local neighborhood sampling is introduced in Section 3.4 to utilize the advantages of hard negative mining [33] and self-paced learning [34] at the same time. This novel sampling method provides to select representative hard negatives while discarding outlier positive examples.

Secondly, the existing metric learning frameworks disregard higher-order relations in data. It may lead to sub-optimal convergence in higher-order solution space. A novel metric learning framework is proposed to alleviate this problem in Section 4.2. At first, input images in the same neighborhood are aligned in a sequential manner by applying trick that is sorting examples in k-nearest neighborhood in ascending order with respect to the selected center. Therefore, RNN which is a powerful deep learning tool to capture long-term dependencies could be exploited to learn embedding space concerning higher-order correlations.

The clustering and retrieval performances of the proposed approach are analyzed and compared with the state-of-the-art methods in two sections. First of all, in Section 3.6, the conventional deep metric learning framework is used to compare introduced approach that consists of presented loss function and sampling method with baseline methods on widely used three public datasets. The superiority of the proposed approach over existing state-of-the-art work is verified on CUB-200-2011, Cars196 and Stanford Online Products datasets by up to 2.0 % on NMI metric, 7.9 % on F_1 score and 5.1 % Recall@1 metric. Besides, these obtained outstanding quantitative results are supported by visual outcomes.

Following that, the introduced deep metric learning framework is compared with the conventional framework on CUB-200-2011, Cars196 and Stanford Online Products datasets for clustering and retrieval tasks in Section 4.3. The efficiency of the proposed framework on modeling higher-order relations in data is validated by quantitative results. It boosts proposed approach that contains a novel loss function and sampling with margin by up to 0.8% on NMI metric, 2.5 % on F_1 score and 1.0 % R@1 metric. Besides, the robustness of the proposed framework to pose and scale variations is demonstrated by qualitative results.

In the future, the proposed approach could be extended in two directions. First, instead of extracting precomputed embeddings from the pre-trained network by utilizing N-pair sampling, the batch construction method that exploits local and global structures of the dataset could be used. Second, it is beneficial to combine proposed approach with deep adversarial metric learning framework [32] to generate synthetic negative examples in absence of dissimilar example in the k-nearest neighborhood.

REFERENCES

[1] H. Oh Song, Y. Xiang, S. Jegelka, and S. Savarese, "Deep metric learning via lifted structured feature embedding," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4004–4012, 2016.

[2] C. Wah, S. Branson, P. Welinder, P. Perona, and S. Belongie, "The caltech-ucsd birds-200-2011 dataset," 2011.

[3] A. Krause and D. Golovin, "Submodular function maximization.," 2014.

[4] "Stanford lecture notes for cs231n." `http://cs231n.stanford.edu`, [Online; accessed 15-August-2018].

[5] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[6] "An intuitive guide to convolutional neural networks." `https://medium.freecodecamp.org`, [Online; accessed 16-August-2018].

[7] "Stanford lecture notes for cs224d." `http://cs224d.stanford.edu`, [Online; accessed 15-August-2018].

[8] L. Van Der Maaten, "Accelerating t-sne using tree-based algorithms," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 3221–3245, 2014.

[9] J. R. Hershey, Z. Chen, J. Le Roux, and S. Watanabe, "Deep clustering: Discriminative embeddings for segmentation and separation," in *Acoustics, Speech and Signal Processing (ICASSP), 2016 IEEE International Conference on*, pp. 31–35, IEEE, 2016.

[10] Z. Liu, D. Wang, and H. Lu, "Stepwise metric promotion for unsupervised video person re-identification," in *Computer Vision (ICCV), 2017 IEEE International Conference on*, pp. 2448–2457, IEEE, 2017.

[11] H.-X. Yu, A. Wu, and W.-S. Zheng, "Cross-view asymmetric metric learning for unsupervised person re-identification," in *IEEE International Conference on Computer Vision*, 2017.

[12] J. Zhou, P. Yu, W. Tang, and Y. Wu, "Efficient online local metric adaptation via negative samples for person reidentification," in *The IEEE International Conference on Computer Vision (ICCV)*, vol. 2, p. 7, 2017.

[13] M. Guillaumin, J. Verbeek, and C. Schmid, "Is that you? metric learning approaches for face identification," in *ICCV 2009-International Conference on Computer Vision*, pp. 498–505, IEEE, 2009.

[14] J. Lu, J. Hu, and Y.-P. Tan, "Discriminative deep metric learning for face and kinship verification," *IEEE Transactions on Image Processing*, vol. 26, no. 9, pp. 4269–4282, 2017.

[15] H. O. Song, S. Jegelka, V. Rathod, and K. Murphy, "Deep metric learning via facility location," in *Computer Vision and Pattern Recognition (CVPR)*, vol. 8, 2017.

[16] A. Choromanska, A. Agarwal, and J. Langford, "Extreme multi class classification," in *NIPS Workshop: eXtreme Classification, submitted*, vol. 1, pp. 2–1, 2013.

[17] I. E.-H. Yen, X. Huang, P. Ravikumar, K. Zhong, and I. Dhillon, "Pd-sparse: A primal and dual sparse approach to extreme multiclass and multilabel classification," in *International Conference on Machine Learning*, pp. 3069–3077, 2016.

[18] B. Mirzasoleiman, A. Badanidiyuru, A. Karbasi, J. Vondrák, and A. Krause, "Lazier than lazy greedy.," in *AAAI*, pp. 1812–1818, 2015.

[19] J. Hu, J. Lu, and Y.-P. Tan, "Deep metric learning for visual tracking," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 26, no. 11, pp. 2056–2068, 2016.

[20] X. Wang, G. Hua, and T. X. Han, "Discriminative tracking by metric learning," in *European conference on computer vision*, pp. 200–214, Springer, 2010.

[21] A. Bellet, A. Habrard, and M. Sebban, “A survey on metric learning for feature vectors and structured data,” *arXiv preprint arXiv:1306.6709*, 2013.

[22] J. V. Davis, B. Kulis, P. Jain, S. Sra, and I. S. Dhillon, “Information-theoretic metric learning,” in *Proceedings of the 24th international conference on Machine learning*, pp. 209–216, ACM, 2007.

[23] K. Q. Weinberger, J. Blitzer, and L. K. Saul, “Distance metric learning for large margin nearest neighbor classification,” in *Advances in neural information processing systems*, pp. 1473–1480, 2006.

[24] S. Ren, K. He, R. Girshick, and J. Sun, “Faster r-cnn: Towards real-time object detection with region proposal networks,” in *Advances in neural information processing systems*, pp. 91–99, 2015.

[25] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, “Going deeper with convolutions,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1–9, 2015.

[26] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” *arXiv preprint arXiv:1409.1556*, 2014.

[27] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” in *Advances in neural information processing systems*, pp. 1097–1105, 2012.

[28] R. Hadsell, S. Chopra, and Y. LeCun, “Dimensionality reduction by learning an invariant mapping,” in *null*, pp. 1735–1742, IEEE, 2006.

[29] F. Schroff, D. Kalenichenko, and J. Philbin, “Facenet: A unified embedding for face recognition and clustering,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 815–823, 2015.

[30] K. Sohn, “Improved deep metric learning with multi-class n-pair loss objective,” in *Advances in Neural Information Processing Systems*, pp. 1857–1865, 2016.

[31] J. Wang, F. Zhou, S. Wen, X. Liu, and Y. Lin, "Deep metric learning with angular loss," in *2017 IEEE International Conference on Computer Vision (ICCV)*, pp. 2612–2620, IEEE, 2017.

[32] Y. Duan, W. Zheng, X. Lin, J. Lu, and J. Zhou, "Deep adversarial metric learning," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2780–2789, 2018.

[33] A. Shrivastava, A. Gupta, and R. Girshick, "Training region-based object detectors with online hard example mining," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 761–769, 2016.

[34] M. P. Kumar, B. Packer, and D. Koller, "Self-paced learning for latent variable models," in *Advances in Neural Information Processing Systems*, pp. 1189–1197, 2010.

[35] R. Xu and D. Wunsch, "Survey of clustering algorithms," *IEEE Transactions on neural networks*, vol. 16, no. 3, pp. 645–678, 2005.

[36] Q. Qian, R. Jin, J. Yi, L. Zhang, and S. Zhu, "Efficient distance metric learning by adaptive sampling and mini-batch stochastic gradient descent (sgd)," *Machine Learning*, vol. 99, no. 3, pp. 353–372, 2015.

[37] S. Boyd, L. Xiao, and A. Mutapcic, "Subgradient methods," *lecture notes of EE392o, Stanford University, Autumn Quarter*, vol. 2004, pp. 2004–2005, 2003.

[38] R. Fletcher and C. M. Reeves, "Function minimization by conjugate gradients," *The computer journal*, vol. 7, no. 2, pp. 149–154, 1964.

[39] J. Goldberger, S. Gordon, and H. Greenspan, "An efficient image similarity measure based on approximations of kl-divergence between two gaussian mixtures," in *null*, p. 487, IEEE, 2003.

[40] J. Lu, G. Wang, and P. Moulin, "Image set classification using holistic multiple order statistics features and localized multi-kernel metric learning," in *Proceedings of the IEEE International Conference on Computer Vision*, pp. 329–336, 2013.

[41] F. Xiong, M. Gou, O. Camps, and M. Sznaier, “Person re-identification using kernel-based metric learning methods,” in *European conference on computer vision*, pp. 1–16, Springer, 2014.

[42] K. Q. Weinberger and G. Tesauro, “Metric learning for kernel regression,” in *Artificial Intelligence and Statistics*, pp. 612–619, 2007.

[43] Z. Feng, R. Jin, and A. Jain, “Large-scale image annotation by efficient and robust kernel metric learning,” in *Proceedings of the IEEE International Conference on Computer Vision*, pp. 1609–1616, 2013.

[44] A. Fix, A. Gruber, E. Boros, and R. Zabih, “A graph cut algorithm for higher-order markov random fields,” in *Computer Vision (ICCV), 2011 IEEE International Conference on*, pp. 1020–1027, IEEE, 2011.

[45] H. Schütze, C. D. Manning, and P. Raghavan, *Introduction to information retrieval*, vol. 39. Cambridge University Press, 2008.

[46] W. S. McCulloch and W. Pitts, “A logical calculus of the ideas immanent in nervous activity,” *The bulletin of mathematical biophysics*, vol. 5, no. 4, pp. 115–133, 1943.

[47] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You only look once: Unified, real-time object detection,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 779–788, 2016.

[48] R. Girshick, “Fast r-cnn,” in *Proceedings of the IEEE international conference on computer vision*, pp. 1440–1448, 2015.

[49] K. He, G. Gkioxari, P. Dollár, and R. Girshick, “Mask r-cnn,” in *Computer Vision (ICCV), 2017 IEEE International Conference on*, pp. 2980–2988, IEEE, 2017.

[50] J. Long, E. Shelhamer, and T. Darrell, “Fully convolutional networks for semantic segmentation,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 3431–3440, 2015.

[51] K. Xu, J. Ba, R. Kiros, K. Cho, A. Courville, R. Salakhudinov, R. Zemel, and Y. Bengio, “Show, attend and tell: Neural image caption generation with visual

attention," in *International conference on machine learning*, pp. 2048–2057, 2015.

[52] O. Vinyals, A. Toshev, S. Bengio, and D. Erhan, "Show and tell: A neural image caption generator," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 3156–3164, 2015.

[53] A. Karpathy and L. Fei-Fei, "Deep visual-semantic alignments for generating image descriptions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 3128–3137, 2015.

[54] Y. Taigman, M. Yang, M. Ranzato, and L. Wolf, "Deepface: Closing the gap to human-level performance in face verification," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1701–1708, 2014.

[55] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *arXiv preprint arXiv:1409.0473*, 2014.

[56] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," *arXiv preprint arXiv:1406.1078*, 2014.

[57] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Advances in neural information processing systems*, pp. 3104–3112, 2014.

[58] D. Bahdanau, J. Chorowski, D. Serdyuk, P. Brakel, and Y. Bengio, "End-to-end attention-based large vocabulary speech recognition," in *Acoustics, Speech and Signal Processing (ICASSP), 2016 IEEE International Conference on*, pp. 4945–4949, IEEE, 2016.

[59] A. Graves, A.-r. Mohamed, and G. Hinton, "Speech recognition with deep recurrent neural networks," in *Acoustics, speech and signal processing (icassp), 2013 ieee international conference on*, pp. 6645–6649, IEEE, 2013.

[60] A. Cauchy, "Méthode générale pour la résolution des systemes d'équations simultanées," *Comp. Rend. Sci. Paris*, vol. 25, no. 1847, pp. 536–538, 1847.

[61] S. Ruder, "An overview of gradient descent optimization algorithms," *arXiv preprint arXiv:1609.04747*, 2016.

[62] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, *et al.*, "Tensorflow: a system for large-scale machine learning.," in *OSDI*, vol. 16, pp. 265–283, 2016.

[63] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," 2017.

[64] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," in *Proceedings of the 22nd ACM international conference on Multimedia*, pp. 675–678, ACM, 2014.

[65] Y. N. Dauphin, R. Pascanu, C. Gulcehre, K. Cho, S. Ganguli, and Y. Bengio, "Identifying and attacking the saddle point problem in high-dimensional nonconvex optimization," in *Advances in neural information processing systems*, pp. 2933–2941, 2014.

[66] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[67] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural computation*, vol. 1, no. 4, pp. 541–551, 1989.

[68] A. Krizhevsky, V. Nair, and G. Hinton, "Cifar-10 (canadian institute for advanced research),"

[69] M. Schuster and K. K. Paliwal, "Bidirectional recurrent neural networks," *IEEE Transactions on Signal Processing*, vol. 45, no. 11, pp. 2673–2681, 1997.

[70] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[71] K.-K. Sung, "Learning and example selection for object and pattern detection," 1996.

[72] S. Ioffe, "Batch renormalization: Towards reducing minibatch dependence in batch-normalized models," in *Advances in Neural Information Processing Systems*, pp. 1945–1953, 2017.

[73] S. Lloyd, "Least squares quantization in pcm," *IEEE transactions on information theory*, vol. 28, no. 2, pp. 129–137, 1982.

[74] H. Jegou, M. Douze, and C. Schmid, "Product quantization for nearest neighbor search," *IEEE transactions on pattern analysis and machine intelligence*, vol. 33, no. 1, pp. 117–128, 2011.

[75] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, *et al.*, "Imagenet large scale visual recognition challenge," *International Journal of Computer Vision*, vol. 115, no. 3, pp. 211–252, 2015.

[76] A. Krogh and J. A. Hertz, "A simple weight decay can improve generalization," in *Advances in neural information processing systems*, pp. 950–957, 1992.

[77] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[78] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in neural information processing systems*, pp. 2672–2680, 2014.

[79] K. Gregor, I. Danihelka, A. Graves, D. J. Rezende, and D. Wierstra, "Draw: A recurrent neural network for image generation," *arXiv preprint arXiv:1502.04623*, 2015.

[80] J. Ba, V. Mnih, and K. Kavukcuoglu, "Multiple object recognition with visual attention," *arXiv preprint arXiv:1412.7755*, 2014.