Locality Sensitive Batch Selection for Triplet Networks

Kyle Martin
School of Computing and Digital Media
Robert Gordon University
Aberdeen, UK
k.martin3@rgu.ac.uk

Nirmalie Wiratunga
School of Computing and Digital Media
Robert Gordon University
Aberdeen, UK
n.wiratunga@rgu.ac.uk

Sadiq Sani
British Telecommunications, PLC
Ipswich, UK
sadiq.sani@bt.com

***Abstract*—Triplet networks are deep metric learners which learn to optimise a feature space using similarity knowledge gained from training on triplets of data simultaneously. The architecture relies on the triplet loss function to optimise its weights based upon the distance between triplet members. Composition of input triplets therefore directly impacts the quality of the learned representations, meaning that a training scheme which optimises their formation is crucial. However, an exhaustive search for the best triplets is prohibitive unless the search for triplets is confined to smaller training regions or batches. Accordingly, current triplet mining approaches use informed selection applied only to a random minibatch, but the resulting view fails to exploit areas of complexity in the feature space. In this work, we introduce a locality-sensitive batching strategy, which uses the locality of examples to create batches as an alternative to the commonly adopted randomly minibatching. Our results demonstrate this method to offer better performance on three image and two text classification tasks with statistical significance. Importantly most of these gains are incrementally realised with as little as 25% of the training iterations.**

***Index Terms*—Deep Metric Learning, Triplet Network, Approximate Nearest Neighbour, Locality Sensitive Hashing, Batch Selection, Self-Paced Learning**

I. Introduction

Deep metric learners are a branch of neural network architectures which use similarity knowledge between input examples to improve representation [1] and create a latent space optimised for similarity-based return [2], [3]. This similarity knowledge is extracted by training on multiple input examples simultaneously. For example, the Triplet Network (TN) is a deep metric learner which learns from three examples concurrently (an anchor, positive and negative example respectively), giving the network its namesake [4]. Throughout training, the network learns to minimise the distance between an anchor and its associated positive example while maximising the distance between an anchor and its associated negative example [4]. Their capability on this task has translated into impressive results on applications such as face recognition/re-identification [5]–[7], image-based search [8] and human activity recognition [9]. Though convergence of these networks can be achieved through creating random triplets, work has shown that a training strategy which optimises triplet creation through active learning can improve training efficiency [5], [8].

Often, these training strategies make use of random minibatches extracted from the training set to offset the complexity of utilising the full set. For example, in [5], the authors 'mine' optimal triplets for network training from within this minibatch by identifying what they describe as semi-hard combinations - i.e. triplets which produce sufficiently large loss to improve weight formation without causing oscillation. Though mining triplets from minibatches offers reduced complexity to methods which target the full training set, it has a key disadvantage. While random minibatches allow an overview of the distribution of the training set, they offer no additional measures to target complex areas such as class boundaries. This is particularly important for triplet networks, because (as with other deep metric learners) their loss is distance-based. We suggest convergence can be achieved faster by considering the locality of examples to inform the creation of minibatches.

In this work we highlight the importance of optimising batch selection before triplet mining approaches are applied. To this end, we propose a novel algorithm, Locality-Sensitive Batching (LSB), which uses locality sensitive methods to focus on example clusters as a substitute for random minibatches as a starting point of further triplet mining. This method can provide the necessary focus on complex class boundary areas to improve training efficiency. In addition, inspired by self-paced learning [10], LSB is able to leverage up to date locality information to inform the creation of training batches. Though locality-sensitive methods can be more expensive than random minibatching, this can be offset by adopting Approximate-Nearest Neighbour (a-NN) methods. In this work we suggest Locality Sensitive Hashing (LSH).

Our findings demonstrate that different batching strategies offer different insights into the space. Training on the full space using a brute force method allows a triplet network to understand the entire distribution of examples, but using the extent of available knowledge quickly becomes expensive. Minibatched strategies provide a randomly sampled overview of the space, but omit potentially useful information about complex areas. On the other hand locality-sensitive batches offer comprehensive focus on a region in the space. However one needs to be aware that focusing in this manner can be detrimental unless an understanding of the full space is also maintained (see Figure 1). Accordingly in this paper we make

978-1-7281-6926-2/20/$31.00 ©2020 IEEE

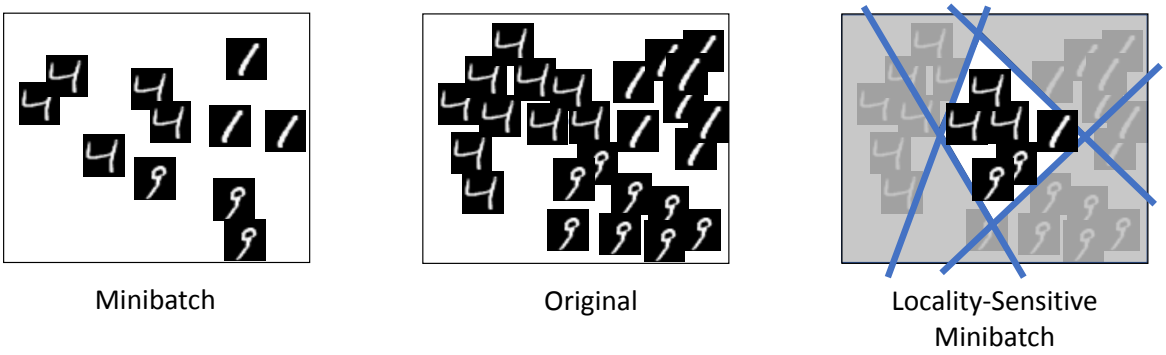

Fig. 1: Visualisation of the different insights offered into the feature space by different training schemes. Left (Minibatch) is created by randomly sampling the original distribution. Right (Locality-Sensitive Minibatch) is created by applying LSB to the original distribution.

the following contributions:

1) We demonstrate that batching strategies are an important design consideration for triplet networks.
2) We present an incremental locality-sensitive batching strategy which allows the batching to evolve alongside example representations over the course of training.
3) Lastly, we present a framework to perform a locality-sensitive batching strategy.

This paper is divided into the following sections: in Section II we discuss related work. In Section III we review the triplet network architecture and identify several different training strategies for our evaluation. In Section IV we present our method for creating locality-sensitive triplets. In Section V we layout the details of our evaluation while in Section VI we discuss the results of our experiments on several datasets. Finally in Section VII we provide some conclusions.

II. Related Work

Other research has shown that sampling is incredibly important in the field of deep metric learning [11]. As the amount of triplet candidates increase in a near cubic manner with the number of examples, it is not feasible to train on all possible combinations. Furthermore, in many situations not every triplet is valuable. Therefore there is much work targeting the optimisation of training triplet networks through sample selection via triplet mining [5], [6], [8].

In [8], the authors use a deep similarity ranking to guide triplet formation for use in learning image similarity. Using a calculated image relevance, they suggest that a relevant but non-matching image should be selected as the negative example and a non-relevant but matching example as the positive example for an anchor image. The authors of [5] expanded upon this idea and removed the concept of relying on an external ranking to decide relevant triplets. They selected pairs by calculating their loss value to preemptively identify their input to the network. They observed that triplets which produced the maximum amount of loss (the 'hardest' triplets) actually caused training to destabilise and network convergence took longer. Instead, focusing on triplets where the distance between the anchor and the negative was greater than the anchor and the positive, but less than the margin (the 'semi-hard' triplets) were more effective for training.

Both of the highlighted approaches operated on a subset of the training set, which was selected randomly from the full distribution, to make computations cheaper. This is consistent with other examples in the literature [6], [7], where authors apply an active learning approach after a subset of the training set has been extracted. We argue that triplet selection actually begins with the selection of that subset, rather than the mining within. Although mining is an important concern, the best triplet cannot be selected if one of the components is not within that initial subset. Identifying this subset is an important aspect of training a triplet network in its own right.

To answer these disadvantages, we are inspired by techniques in Case-Based Reasoning (CBR) to create a locality-based batching method which is not reliant on class information or previously trained models and iteratively updates in response to latest network output. Specifically we highlight work which leverages similarity knowledge to cluster the case-base and reduce the complexity of case retrieval. Examples include cluster-based retrieval from large-scale case-bases of image [12], text [13] and simulation data [14]. Research in this field has established that the coverage knowledge generated by clustering approaches can be exploited for sample selection. For example, in [15] the authors use clustering methods to identify the most important cases for labeling from an unlabeled set. We are motivated by these findings to develop a batch selection technique. However clustering can be an expensive process, thus we explore a method to approximate it using an approximate-Nearest Neighbour algorithm (a-NN).

A-NNs are a set of techniques to inexpensively perform neighbourhood computations on large sets of examples. They offer a means to extract similar examples from a dataset at a fraction of the cost of brute-force nearest neighbour methods, with the drawback of usually being less accurate [16], [17]. In particular, in this work we suggest the use of Locality-Sensitive Hashing (LSH). LSH is a data independent a-NN method to economically estimate nearest neighbour computations by randomly dividing the feature space into distinct areas known as 'buckets', which preserve locality knowledge from the original space [18]. When a query is presented,

it is indexed into a bucket. Similarity metrics are therefore performed only between a query and the contents of the relevant bucket to establish similarity knowledge in that neighbourhood. Though indexing examples offers a means to decrease the complexity of pairwise similarity computations such that we could consider using informed selection on the full space, we suggest that this is unnecessary. Instead, we can use the buckets created in the initial stages of LSH as a minibatch for input to the network. More details on this process is available in Section IV.

There has been some previous work in using clustering techniques to inform network training by altering the triplet loss function [19], [20]. However, these methods typically require *a priori* knowledge [19], or offer reduced flexibility to incorporate other training methods (such as hard sample mining) because the clustering mechanism is tightly coupled to loss calculation [20]. It would be desirable to have a solution with no previous knowledge requirements, and which could function as part of an ecosystem of training methods.

We are aware of only limited work which directly targets minibatching of instances before input to the network [21]. In this work, the authors process images of faces through a classifier to improve representations, before passing those learned representations to a k-means algorithm to perform clustering for initial batch selection. However, this approach presents several disadvantages. Most importantly, it is a learned clustering method with corresponding overhead and dependence upon access to labelled data. This is problematic, as triplet networks perform best in situations where labelled data is scarce or totally unavailable. The process is also not iterative as clustering is performed on the output of the classification model at the start of training and remains static. Ideally, locality-based minibatching should exploit the latest network output to ensure its batches are relevant to the network at that point in training. In response and inspired by research in curriculum learning [22], in particular self-paced learning [10], the proposed method in this paper uses latest network output to inform its clustering. The result is that the batches of input data created by LSB are based upon an up to date representation of the latent space, and as such faithfully represent current 'difficult' potential triplets for network input.

III. Triplet Networks

Triplet networks are deep metric learners which learn from three input examples simultaneously. These inputs are the anchor example (x^a), a positive example (x^+) and a negative example (x^-), which together are described as a triplet. The anchor example acts as a point of comparison, meaning that the positive and negative examples are dictated by their relationship to the anchor (i.e. matching and not matching respectively). The goal of training is to create a space optimised for similarity-based return by minimising the distance between an anchor and its associated positive example while maximising the distance between an anchor and its associated negative example.

A triplet network is comprised of three identical 'sub-networks' (see Figure 2). Typically a deep learning architecture, which can be as shallow or as deep as necessary. Each sub-network creates an embedding for one input (i.e. an individual member of the triplet) before the error is calculated using triplet loss.

A. Training a Triplet Network

Let us introduce the notation used in this paper. Let $\mathcal{X}$ be a set of labeled examples, such that example, $x \in \mathcal{X}$ and $y(x)$ is a function that returns the class label, y, of x. In the context of this paper, we will define matching examples as those which have the same class ($y(x^+) = y(x^a)$) while non-matching examples will have differing classes ($y(x^-) \neq y(x^a)$). The embedding function θ is a parametric model of any one of the identical sub-networks creating the learned representation of a given x. We can then represent triplet loss L as so:

$$L = max(0, (D_W(\theta(x^a), \theta(x^+)) - D_W(\theta(x^a), \theta(x^-)) + \alpha)) \quad (1)$$

Where D_W is a function to calculate the distance between two embeddings and α is the margin which must exist between an anchor and negative example. This formula will generate a loss value in situations where the anchor example is closer to the negative example than it is to the positive example. The network is therefore penalised until similar cases are placed closer together in the feature space. The $max()$ function ensures that only loss values greater than zero impact network weights.

However, there are some issues with this. As the network approaches convergence, random formation of triplets has an increased likelihood to provide triplets which will generate a loss of zero. This is because the feature space will be approaching optima. The result is the network will train for increasing periods of time with decreasing improvements to its weights; hence the importance of sample selection and training optimisation. Simply put, we want to form triplets which will maximise loss for the improvement of the network and allow it to converge towards optima more quickly.

B. Creating Triplets

Let us first identify a baseline algorithm for randomly creating triplets from the full training set, where $\mathcal{T}()$ is a function to create a triplet.

In Algorithm 1, $is\_matching()$ is a function which separates $\mathcal{X}$ (or any subset) into two sets, $\mathcal{P}os$ and $\mathcal{N}eg$, based on each member's relationship to a given anchor example x^a. Members of $\mathcal{P}os$ have a matching class label with x^a and $\mathcal{N}eg$ have a non-matching class label. Note that this function could be adapted to be non-class reliant. Algorithm 1 is relatively inexpensive to perform, but as mentioned above, there is no guarantee that the created triplets will result in any loss for the network. This is a problem which gets worse over the course of training as the optimal representation of the space is approached.

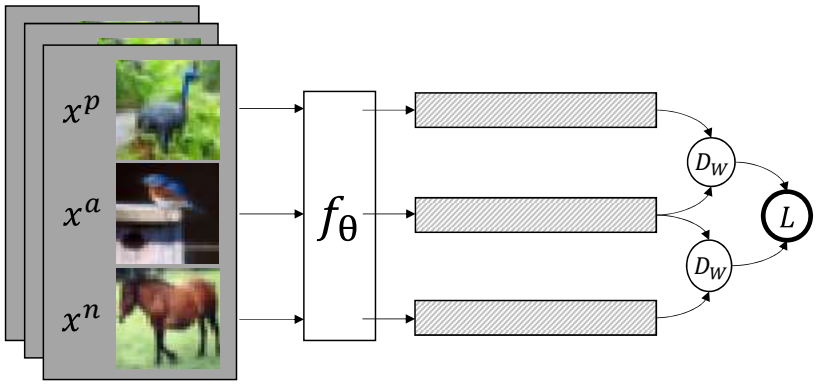

Fig. 2: Batched triplet network training on the CIFAR10 dataset. The representations learned by each sub-network for each input image are improved over time by using knowledge around the relationship between inputs during training.

Algorithm 1: Create random triplets from full training set.

Random: $\mathcal{X}_{RND}(n)$
for x^a *in* $\mathcal{X}$ **do**
  $\mathcal{P}os, \mathcal{N}eg = is\_matching(x^a, \mathcal{X})$
  x^+ := rnd_selection($\mathcal{P}os$)
  x^- := rnd_selection($\mathcal{N}eg$)
  $\mathcal{T}_i := \mathcal{T}(x^a, x^+, x^-)$
  $\mathcal{T} = \mathcal{T} \cup \mathcal{T}_i$
end
return $\mathcal{T}$

Algorithm 2: Develop random triplets from a mini-batch.

Random Minibatch: $\mathcal{X}_{MR}(n)$
$\mathcal{M} = \mathcal{M}(\mathcal{X})$
for $m_i \ldots \mathcal{M}$ **do**
  for x^a *in* m_i **do**
    $\mathcal{P}os, \mathcal{N}eg = is\_matching(x^a, m_i)$
    x^+ := rnd_selection($\mathcal{P}os$)
    x^- := rnd_selection($\mathcal{N}eg$)
    $\mathcal{T}_i := \mathcal{T}(x^a, x^+, x^-)$
    $\mathcal{T} = \mathcal{T} \cup \mathcal{T}_i$
  end
end
return $\mathcal{T}$

To counter these issues, there must be some concept of identifying triplets which are meaningful for training - an informed approach. However, checking every example is too expensive. Hence the importance of extracting minibatches. By examining minibatches of the data at a time, the complexity of an informed approach is considerably reduced. The question then becomes how best to identify subsets of the data which lend themselves to minibatches. In the literature [5], [7], [8], authors suggest randomly sampling to get an overview of the space. We suggest that the batching method is an important design consideration, and by selecting the appropriate method significant improvements can be made.

C. Creating Triplets from a Random Minibatch (MR)

Hereon, we use the term 'minibatch' to refer to any subset of $\mathcal{X}$. Minibatches which are representative of the original space can be easily created using Algorithm 2. This can be trivially adapted to ensure stratification in cases of class imbalance.

In Algorithm 2 m is a minibatch of examples from $\mathcal{X}$, such that $m \subset \mathcal{X}$. $\mathcal{M}$ is then the complete set of minibatches and the function $\mathcal{M}()$ creates a set of minibatches from within $\mathcal{X}$.

Though training using minibatches of examples in this way reduces the potential number of triplets (thereby reducing pairwise similarity computations and complexity), it does not provide any focus on complex areas of the feature space. This is because the random selection of the minibatch allows it to be representative of the data distribution as a whole, without allowing any room for specific localised knowledge of the feature space. Clustering methods offer potential to fill this gap, but are difficult to justify due to their high initial resource requirements. In the next section, we will describe how we adopted methods from locality-sensitive hashing to inform the creation of clusters.

IV. Locality-Sensitive Batching (LSB)

It is clear that the greater the loss generated by a given input, the greater its contribution to the network weights (hence the intuition behind [5]). Because triplet networks utilise a distance-based loss, the contribution of a given triplet is decided by the distance between its constituent members. With that in mind, the most appropriate triplet for a given anchor is likely to exist within the same locality. In this section, we detail how we adapt Locality-Sensitive Hashing (LSH) to develop locality-sensitive batches for network training.

A. Locality-Sensitive Hashing (LSH)

LSH defines a family of algorithms which use locality-sensitive hashing functions to cluster information These hashing functions are described as locality-sensitive because there

is high probability that similar instances share the same function, but low probability that dissimilar instances share that function:

$$P\big(h(x_i) == h(x_j)\big) \begin{cases} \text{High, if } D_W(x_i, x_j) \text{ is Low} \\ \text{Low, if } D_W(x_i, x_j) \text{ is High} \end{cases} \quad (2)$$

In Equation 2, $P()$ is a probability function, $h()$ is a hashing function and x_i and x_j are arbitrary examples from within a dataset $\mathcal{X}$. By hashing the space in this manner the complexity of identifying an example's locality becomes sub-linear.

To achieve this in its simplest form, LSH uses random projections to hash the feature space. This method divides the feature space into separate 'buckets' by partitioning the space using a configurable number of random divisions called 'projections'. Every example in the dataset is indexed into a bucket and empty buckets are discarded. To formalise this process, let us consider a random projection, v, with the same dimensionality as x. Since there is more than one projection into the space, v belongs to an ordered set V. To index each example in a dataset, $x \in \mathcal{X}$, we identify the relevant bucket by calculating a hash key, H, formed from a series of binary values, h_i, which are indicative of that example's relationship to each projection. Effectively, this process identifies 'which side of the projection' the example inhabits within the space and are calculated by placing a threshold on a dot product comparison:

$$h_i = \begin{cases} 0, \text{ if } x \cdot v_i < 0 \\ 1, \text{ if } x \cdot v_i >= 0 \end{cases} \quad \forall v_i \in V \quad (3)$$

H is then the ordered concatenation of each h_i it contains, allowing it to act as an identifier for a specific bucket. As the indexing system is based upon a similarity comparison, the buckets preserve locality information from the original distribution of the space and there is a high probability that similar examples are allocated to the same bucket (supporting the declaration in Equation 2). These buckets are therefore well-placed to quickly identify the locality of an instance, something we will exploit in the following subsection.

B. Locality-Sensitive Batching (LSB)

In the literature, LSH is often used to reduce complexity for similarity comparisons [16]. However, each bucket can also effectively be considered as a cluster. We have found that the these clusters offer a good alternative form of batch selection to random minibatching with only trivial adaptation (see Lines 1-11 of Algorithm 3). Furthermore, we ensure that the information from clustering is up to date by basing our locality informed clusters on the latest network output, θ. The intuition is that this will allow the network to maintain focus on complex areas which are most relevant to its current parameters. Hence, on the first epoch of the network the input data is split into batches by using LSH on the original data representation (i.e. $LSH(\mathcal{X})$), while in all subsequent epochs input data is batched by using LSH on the network output (i.e. $LSH(\theta(\mathcal{X}))$). We make this distinction as the network is initialised with random weights and so performing LSH on its output before any training has occurred would not produce meaningful batches.

In Algorithm 3, LSH is a function to extract a set of locality-sensitive buckets from $\mathcal{X}$ and b is an individual bucket from within $\mathcal{B}$, such that $b \in \mathcal{B}$. Finally, $pure()$ is a function which returns True if the selected bucket contains only a single class (or False otherwise) and R is an empty set which is eventually populated with anchor cases which exist in pure clusters or as the sole member of a cluster. Naturally, as the network converges we expect the number of impure buckets to decrease (see Figure 3).

Algorithm 3: Develop random triplets from a bucket.

1 **Locality-Sensitive Batching:** $\mathcal{X}_{LSB}(n)$
2 **if** *epoch = 1* **then**
3 | $\mathcal{B}$ = LSH($\mathcal{X}$)
4 **else**
5 | $\mathcal{B} = LSH(\theta(\mathcal{X}))$
6 $R = \emptyset$
7 **for** $b_i \ldots \mathcal{B}$ **do**
8 | **if** *not pure(b_i)* **then**
9 | | **for** x^a *in* b_i **do**
10 | | | $\mathcal{P}os, \mathcal{N}eg = is\_matching(x^a, b_i)$
11 | | | x^+ := rnd_selection($\mathcal{P}os$)
12 | | | x^- := rnd_selection($\mathcal{N}eg$)
13 | | | $\mathcal{T}_i$:= $\mathcal{T}(x^a, x^+, x^-)$
14 | | | $\mathcal{T} = \mathcal{T} \cup \mathcal{T}_i$
15 | | **end**
16 | **else**
17 | | $R := R \cup b_i$
18 **end**
19 **for** $x^a \ldots R$ **do**
20 | $\mathcal{P}os, \mathcal{N}eg = is\_matching(x^a, R)$
21 | x^+ := rnd_selection($\mathcal{P}os$)
22 | x^- := rnd_selection($\mathcal{N}eg$)
23 | $\mathcal{T}_i$:= $\mathcal{T}(x^a, x^+, x^-)$
24 | $\mathcal{T} = \mathcal{T} \cup \mathcal{T}_i$
25 **end**
26 $shuffle(\mathcal{T})$
27 **return** $\mathcal{T}$

As buckets are created using locality knowledge, feeding triplets into the network can enforce sequential learning. This in turn can be problematic, because the implicit curriculum could be non-optimal. With that in mind, we need to randomise the order of the triplets to allow an understanding of the overall distribution of the space. We do this in two ways. Firstly, if we fail to identify a cluster for a given example (or if the cluster identified is 'pure'), we randomly combine it with other examples where a cluster could not be identified to create a triplet. Secondly, we input the triplets we have gained from our buckets to the network in a random order (see Lines 16-24 of Algorithm 3).

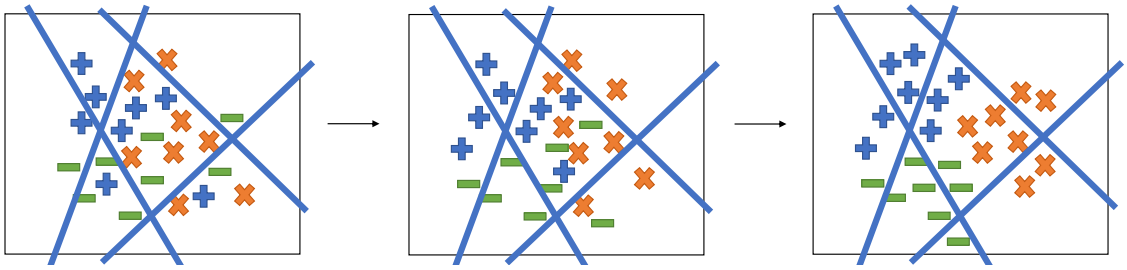

Fig. 3: Distribution of examples in buckets throughout training. As training progresses the number of impure buckets decreases.

V. Evaluation

In this section, we offer details of our evaluation of the proposed method. The goal of our evaluation is to demonstrate that different batching strategies can have a profound effect on the quality of representations gained from DMLs. We do not compare against triplet mining methods, as these are a subsequent step to a batching strategy. We create a comparison using our proposed method, locality-sensitive batching, and the most popular strategy from literature, minibatched random:

1) **Minibatched Random** (MR): Triplets are randomly generated from within a minibatch of the training set. Minibatches are distinct and contain non-overlapping examples. This algorithm will act as our baseline for comparison.
2) **LSB Random** (LSB): Triplets are randomly generated from within a local neighbourhood of each anchor case in the training set. These neighbourhoods are distinct and contain non-overlapping examples. Anchors which have a 'pure' neighbourhood are randomly combined to create triplets.

In both instances, we ensure that every example in the training set is utilised as an anchor only once per training epoch. This means that there are as many triplets per training epoch as there are examples in the training set.

We assess the quality of learned representations using a similarity-based return task. To achieve this, we perform an empirical comparison of the representations gained from each training scheme using k-NN accuracy as a proxy for representation goodness. Classification is performed using k-NN, where $k = 3$ and similarity is measured using cosine similarity. We perform a one-tail t-test to establish statistical significance at a confidence level of 95% on classification accuracy from network output. We also examine each algorithms' capacity to learn over time by comparing averaged accuracy on each test set for increasing number of training epochs. This is important because improvements that LSB offers are likely to be in the form of training efficiency.

A. Network Architecture

All architectures used ReLU activations and the Adam optimizer [23] and produced an output representation of the size 128. For all other variables, including number of batches for networks using MR and number of projections for networks using LSB, we implemented an empirical evaluation to identify the best performing hyperparameters for each dataset (see Table I). Note that since projections in LSH are random, the number of buckets can vary between runs. This is because there is potential to create empty buckets which are discarded. Therefore, it is more suitable to maintain the number of projections as constant. Batch sizes were set such that they were a multiple of the number of labels contained in each dataset (i.e in MNIST there are 10 classes, so the batch size was a multiple of 10).

B. Datasets

Below are some of the details for each of the datasets used to evaluate the methods presented in this paper. We have tested this algorithm on three image (MNIST, CIFAR10 and STL-10) and two text (IMDB, REUTERS) classification tasks. We selected numerous datasets across different classification tasks to display the overall versatility and utility of the proposed batching method. In all situations, 5-fold cross-validation was used to create distinct train and test sets. This was because we were not aiming for state-of-the-art, but to empirically demonstrate that different batching methods can impact network performance to a statistically significant degree.

1) Image Classification Datasets: We have selected 3 popular image classification datasets from the literature. These datasets were selected because of the triplet network's utility in image-based search and to demonstrate our algorithm as applicable in this domain. We did not use data augmentation in any case, as our goal was merely to compare the two batching methods.

MNIST is a handwriting recognition dataset comprised of 70,000 greyscale images of handwritten single-digit numbers. Images are 28×28 pixels and have one of ten classes (the numbers zero to nine).

CIFAR10 [24] is an object recognition dataset comprised of 60,000 colour images split evenly between 10 classes. Each image is 32×32 pixels in size and features one of ten distinct objects that are used to identify its class label (airplane, automobile, bird, cat, deer, dog, frog, horse, ship or truck).

STL-10 [25] is an object recognition dataset comprised of 13,000 labelled and 100,000 unlabelled images extracted from ImageNet. We only utilise the labelled images in our experiments, which are divided evenly between 10 classes (airplane, bird, car, cat, deer, dog, horse, monkey, ship, truck). The images are 96×96 pixels in size which is substantially larger than examples from CIFAR10 and making it a challenging

Dataset	No. of Features	Sub-Network (Layers)	Kernel	Epochs	Minibatches	Batch Size	Projections
MNIST	784	MLP (3 Dense)	-	10	1000	50	18
CIFAR10	32x32	CNN (4 Conv., 2 Dense)	(3, 3)	100	1000	40	18
STL-10	96x96	CNN (4 Conv., 2 Dense)	(3, 3)	100	250	40	6
IMDB	5000	MLP (3 Dense)	-	20	1000	40	12
REUTERS	1000	MLP (2 Dense)	-	10	90	138	5

TABLE I: Summary of relevant network hyperparameters

Classification	Dataset	Method	Accuracy throughout Training (%)			
			25%	50%	75%	100%
Image	MNIST	MR	95.98	96.66	96.95	96.99
		LSB	95.81	96.93	97.19	**97.40***
	CIFAR10	MR	57.81	64.41	66.55	66.68
		LSB	62.65*	66.68*	68.44*	**69.02***
	STL-10	MR	50.26	57.79	60.76	**61.67**
		LSB	50.97	61.16*	61.51*	61.63
Text	IMDB	MR	85.86	87.39	87.90	88.04
		LSB	87.75*	87.80	88.30*	**88.33**
	REUTERS	MR	74.42	75.01	76.02	76.47
		LSB	77.13*	78.03*	78.32*	**78.68***

TABLE II: Summary of algorithm performance throughout training

benchmark to test the scalability of our proposed method, as well as much closer to the size of commercial images (such as the image of a product on its Amazon web page).

2) Text Classification Datasets: All text classification datasets were preprocessed by removing stop-words and stemming words to their root form. Term-frequency/inverse-document-frequency (tf-idf) was used to generate a numerical representation for input to our network, using the 5,000 most common words in the case of IMDB and the 1,000 most common words in the case of Reuters.

The Large Movie Review Dataset [26] is comprised of 50,000 labeled film reviews scraped from the Internet Movie Database (IMDB). Polarized reviews have been extracted and labeled as either 'positive' (where a review score is greater than 6) or 'negative' (where a review score is lower than 4) to create a binary sentiment analysis task. Though the dataset also contains a large number of unlabeled reviews, we did not use these. We selected the IMDB dataset as its boundary is naturally complex due to the presence of both concept complexity and subjective judgment.

The Reuters dataset is a document classification dataset comprised of structured news wire articles. We used the ModApte subset of the Reuters-21578 benchmark, which contains 11,228 documents each given one of 46 labels. Reuters was selected as it is particularly challenging for minibatch approaches, given its inherent data imbalance. In this dataset it is impossible for minibatches to be non-overlapping due to the very few examples that are associated with some labels. This issue does not apply to locality-sensitive approaches, as no constraints are placed on their contents.

VI. Results

The results for each dataset appear in Table II with bold font used to indicate the highest achieved accuracy for a dataset and asterisks indicating performance which is better than the baseline with statistical significance at 95% confidence. As can be observed, LSB outperforms the baseline on all tested datasets with significance for at least some portion of training. On 3 of the 5 tested datasets (MNIST, CIFAR10 and Reuters), LSB achieves a statistically significant improvement on accuracy at the end of training. On every tested dataset LSB converges to optima faster, approximating the baseline performance with only 50% of the required training or less.

On MNIST, though differences are less pronounced during early training, our approach does converge to a statistically significant higher accuracy. The advantages of LSB can be seen on CIFAR10, where it outperforms MR from very early in training, achieving performance improvements that are statistically significant from 10% of training onwards.

Though it would seem from Table II that the baseline for STL-10 converges to the same accuracy as LSB, we actually converge to optima much earlier in training. By 50 epochs, the accuracy achieved by LSB is already at 61.16%, which is a 4% improvement over MR at the same number of epochs.

On the IMDB dataset, we observe that LSB has great benefits very early in training, with less improvements as time goes on. This is because LSB can focus on the complex boundary cases that are difficult to classify. The similar performance achieved is indicative of the difficulty to wholly separate the positive and negative viewpoints in this task.

It is interesting to note the superior performance of LSB on the Reuters dataset. MR struggles when faced with many classes or imbalanced data. This is not a problem for LSB, as the method is only concerned with a small neighbourhood of the space. Thus we suggest these results seem indicative that LSB is more suitable in problems with class imbalance.

VII. Conclusions

In conclusion, we have demonstrated a locality-sensitive minibatching method, LSB, which utilises locality information to inform selection of minibatches for training a triplet network. The networks trained using LSB obtained better accuracy than random minibatching methods on our evaluation task, suggesting that locality-sensitive minibatches are a better starting point for further active learning approaches. This is indicative that the method by which to obtain a minibatch for triplet mining is an important concern for the training of triplet networks. The value that is added by LSB emphasises the need for this as a design consideration when training a triplet network.

In future work, we aim to further explore how clustering methods can improve training of deep metric learners. In particular, we will expand our investigation to explore the correlation between cluster sizes and training speed. This will offer a more thorough understanding of the impact that locality information can have on training strategies for DMLs. Furthermore, a drawback of the suggested approach is the reliance on identifying an appropriate number of projections to split the space. We hope to automate this in future.

References

[1] J. Bromley, I. Guyon, and Y. LeCun, "Signature verification using a 'siamese' time delay neural network," *International Journal of Pattern Recognition and Artificial Intelligence*, vol. 7, no. 4, pp. 669 – 688, August 1993.

[2] S. Chopra, R. Hadsell, and Y. LeCun, "Learning a similarity metric discriminatively, with application to face verification," in *Proc. of the 2005 IEEE Comp. Soc. Conf. on Computer Vision and Pattern Recognition*, ser. CVPR '05. Washington, DC, USA: IEEE Computer Society, June 2005, pp. 539 – 546.

[3] G. Koch, R. Zemel, and R. Salakhutdinov, "Siamese neural networks for one-shot image recognition," in *Deep Learning Workshop*, ser. ICML '15, July 2015.

[4] E. Hoffer and N. Ailon, "Deep metric learning using triplet network," in *Similarity-Based Pattern Recognition*, A. Feragen, M. Pelillo, and M. Loog, Eds. Cham: Springer International Publishing, 2015, pp. 84 – 92.

[5] F. Schroff, D. Kalenichenko, and J. Philbin, "Facenet: A unified embedding for face recognition and clustering," *CoRR*, vol. abs/1503.03832, 2015.

[6] W. Liao, M. Ying Yang, N. Zhan, and B. Rosenhahn, "Triplet-based deep similarity learning for person re-identification," in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 385–393.

[7] A. Hermans, L. Beyer, and B. Leibe, "In defense of the triplet loss for person re-identification," *arXiv preprint arXiv:1703.07737*, 2017.

[8] J. Wang, Y. Song, T. Leung, C. Rosenberg, J. Wang, J. Philbin, B. Chen, and Y. Wu, "Learning fine-grained image similarity with deep ranking," in *Proc. of the 2014 IEEE Conf. on Computer Vision and Pattern Recognition*, ser. CVPR '14. Washington, DC, USA: IEEE Computer Society, June 2014, pp. 1386 – 1393.

[9] K. Martin, A. Wijekoon, and N. Wiratunga, "Human activity recognition with deep metric learners." ICCBR 2019 Workshop Proceedings, 2020.

[10] M. P. Kumar, B. Packer, and D. Koller, "Self-paced learning for latent variable models," in *Advances in Neural Information Processing Systems 23*, ser. NIPS '10. Red Hook, NY, USA: Curran Associates, Inc., December 2010, pp. 1189 – 1197.

[11] C.-Y. Wu, R. Manmatha, A. J. Smola, and P. Krahenbuhl, "Sampling matters in deep embedding learning," in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 2840–2848.

[12] Yixin Chen, J. Z. Wang, and R. Krovetz, "Clue: cluster-based retrieval of images by unsupervised learning," *IEEE Transactions on Image Processing*, vol. 14, no. 8, pp. 1187–1201, Aug 2005.

[13] I. S. Altingovde, R. Ozcan, H. C. Ocalan, F. Can, and O. Ulusoy, "Large-scale cluster-based retrieval experiments on turkish texts," in *Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, ser. SIGIR '07. New York, NY, USA: ACM, 2007, pp. 891–892. [Online]. Available: http://doi.acm.org/10.1145/1277741.1277961

[14] M. R. Lucca, A. G. L. Junior, E. P. Freitas, and L. A. Silva, "A case-based reasoning and clustering framework for the development of intelligent agents in simulation systems," in *The Thirty-First International Flairs Conference*, 2018.

[15] N. Wiratunga, S. Craw, and S. Massie, "Index driven selective sampling for cbr," in *Proceedings of the Fifth International Conference on Case-Based Reasoning*. Springer, 2003, pp. 637–651.

[16] M. Muja and D. G. Lowe, "Fast approximate nearest neighbors with automatic algorithm configuration." *VISAPP (1)*, vol. 2, no. 331-340, p. 2, 2009.

[17] B. Kulis and K. Grauman, "Kernelized locality-sensitive hashing for scalable image search," in *Computer Vision, 2009 IEEE 12th International Conference on*. IEEE, 2009, pp. 2130–2137.

[18] M. Slaney and M. Casey, "Locality-sensitive hashing for finding nearest neighbors [lecture notes]," *IEEE Signal processing magazine*, vol. 25, no. 2, pp. 128–131, 2008.

[19] W. Ge, W. Huang, D. Dong, and M. R. Scott, "Deep metric learning with hierarchical triplet loss," *CoRR*, vol. abs/1810.06951, 2018. [Online]. Available: http://arxiv.org/abs/1810.06951

[20] O. Rippel, M. Paluri, P. Dollar, and L. Bourdev, "Metric learning with adaptive density discrimination," 2015.

[21] C. Wang, X. Zhang, and X. Lan, "How to train triplet networks with 100k identities?" in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 1907–1915.

[22] Y. Bengio, J. Louradour, R. Collobert, and J. Weston, "Curriculum learning," in *Proc. of the 26th Annual International Conf. on Machine Learning*, ser. ICML '09. New York, NY, USA: ACM, June 2009, pp. 41 – 48.

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

[24] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," Citeseer, Tech. Rep., 2009.

[25] A. Coates, A. Ng, and H. Lee, "An analysis of single-layer networks in unsupervised feature learning," in *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, ser. Proceedings of Machine Learning Research, G. Gordon, D. Dunson, and M. DudÃk, Eds., vol. 15. Fort Lauderdale, FL, USA: PMLR, April 2011, pp. 215–223.

[26] A. L. Maas, R. E. Daly, P. T. Pham, D. Huang, A. Y. Ng, and C. Potts, "Learning word vectors for sentiment analysis," in *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies - Volume 1*, ser. HLT '11. Stroudsburg, PA, USA: Association for Computational Linguistics, 2011, pp. 142–150.