Informed pair selection for self-paced metric learning in Siamese neural networks.

MARTIN, K., WIRATUNGA, N., MASSIE, S. and CLOS, J.

2018

Informed Pair Selection for Self-Paced Metric Learning in Siamese Neural Networks

Kyle Martin[0000−0003−0941−3111] ✉, Nirmalie Wiratunga[0000−0003−4040−2496], Stewart Massie[0000−0002−5278−4009], Jérémie Clos[0000−0003−4280−5993]

Robert Gordon University, Aberdeen, Scotland
{k.martin, n.wiratunga, s.massie, j.clos} @rgu.ac.uk

Abstract. Siamese Neural Networks (SNNs) are deep metric learners that use paired instance comparisons to learn similarity. The neural feature maps learnt in this way provide useful representations for classification tasks. Learning in SNNs is not reliant on explicit class knowledge; instead they require knowledge about the relationship between pairs. Though often ignored, we have found that appropriate pair selection is crucial to maximising training efficiency, particularly in scenarios where examples are limited. In this paper, we study the role of informed pair selection and propose a 2-phased strategy of exploration and exploitation. Random sampling provides the needed coverage for exploration, while areas of uncertainty modeled by neighbourhood properties of the pairs drive exploitation. We adopt curriculum learning to organise the ordering of pairs at training time using similarity knowledge as a heuristic for pair sorting. The results of our experimental evaluation show that these strategies are key to optimising training.

Keywords: Deep Learning · Siamese Neural Networks · Active Learning · Case-Based Reasoning · Machine Learning · Metric Learning

1 Introduction

The Siamese Neural Network (SNN) is a deep learning architecture which trains upon pairs of input data to learn a metric space in which training instances can be placed. The expectation of paired learning is that the learned space can better represent salient relationships between pairs which can then be better captured in Euclidean space. Originally used in binary classification tasks such as signature verification [2] and face recognition [3], SNNs have recently been generalised to multi-class classification [6].

Central to SNN learning is the use of similarity knowledge to gauge closeness of data points such that it provides a meaningful matching criterion. The expected outcome of SNN training is to have instances deemed similar to be mapped closer together. As SNNs are metric learners that develop a new representation of the original data, in a classification setting they require a non-parametric learner (such as k-NN) to perform the explicit classification. However, a benefit of SNNs is that they do not require full class knowledge during training, as they are learning on the basis of whether pairs meet the matching criteria, not whether they belong to a specific label. For this reason, recent work has demonstrated these networks can perform effectively even when working with extremely limited training data, such as in a one-shot learning environment [6].

Surprisingly, pair selection and ordering have been given little attention in SNNs, despite showing promising results in the closely-related Triplet Networks (TNs) [17, 20]. Pair selection and ordering directly informs the data relationships that the network will learn and be trained upon, thereby directly influencing the metric developed by the network. It is our view that these strategies play an important role in both improving training time and ensuring high overall performance of SNNs. Pair selection and presentation order are likely to be particularly relevant in data-budgeted scenarios where there are only a small number of annotated examples.

In this paper we discuss the importance of pair selection strategies and their effect on training. We are of the opinion that only by understanding the impact of sample selection in multiple-input networks can we build upon these ideas for application towards recent advancements in deep metric learners. For this reason we start with SNNs. Although we limit our scope to SNNs, the contribution from this paper is applicable to other neural network architectures that learn from multiple examples - particularly triplet [5] and matching networks [19].

The contributions of this paper are four fold: (1) we demonstrate the importance of having a pair selection strategy in SNNs; and (2) we introduce several pair selection strategies, including methods for informed pair selection that optimise pair creation using explorative and exploitative strategies. Taking inspiration from self-paced learning; (3) we demonstrate that pair ordering can improve SNN network performance; and (4) we introduce a pair complexity heuristic for ordering that draws on knowledge about the neighbourhood properties of pairs[1].

This paper is organised as follows: in Section 2 we explore work related to pair selection and curriculum learning; in Section 3 we formalise pair creation and its role in SNN learning before we introduce the concepts of pair ordering and informed pair selection employed in our work; a comparative study of proposed methods appear in Section 4 with results on the MNIST, Large Movie Review Dataset (LMRD) and SelfBACK [2] datasets appearing in Section 5; and finally conclusions in Section 6.

2 Related Work

Employing informed selection for training data is increasingly gaining attention for deep learning architectures. Both optimising for batch size and the order in which training examples are processed have shown to be effective for achieving performance improvements [1, 10]. Typically network loss is exploited to create ranking heuristics with significant speed-up gains observed when processing harder examples first. Training on an increasing ratio of 'hard' samples can be seen as adopting an 'exploitation' strategy where focus is maintained on known 'hard' problems. It is interesting to note that meta-learning strategies, such as boosting, do precisely this with weak learners; whereby

[1]The code associated with this paper is publicly accessible from https://github.com/RGU-AI/Informed-Pair-Selection

[2]The SelfBACK project is funded by European Union's H2020 research and innovation programme under grant agreement No. 689043. More details available: http://www.selfback.eu. The SelfBACK dataset associated with this paper is publicly accessible from https://github.com/selfback/activity-recognition

model learning is focused on examples that were incorrectly solved previously [18]. However such a strategy alone in the context of informed sample selection can be detrimental, if 'exploration' of the space of possible problems is ignored [10]. We study how both exploration and exploitation strategies can be utilised for informed pair selection. Specifically we consider budgeted learning scenarios associated with learning an embedding function [9], where it has been shown that picking more suitable examples will return greater results in circumstances where labeled data is limited [4].

Paired examples in relation to triplet networks (TNs) [5] help learn useful feature embeddings (representations) by distance comparisons [20, 17]. Like SNNs, the goal of training is to develop an embedding function which minimises distance between the positive examples and the query examples, while maximising the distance between the negative example and the query. Unlike with SNNs, TNs form a triplet instance from a negative and positive pair given a query. Heuristics that are static (neither exploratory or exploitative) based on initial similarity (relevance) alone were found to perform poorly [20]. Using heuristics that continually update to reflect the triplets the network is likely to find difficult in the next iteration, such as exploiting according to the loss value, was found to give superior performance [17]. However, loss information is not available from the start of training so the network must complete an initial 'dry run' to retrieve this information. In our work we consider how heuristics that utilise similarity knowledge calculated from the most recent network embedding can contribute towards formulating a more dynamic ranking heuristic for training examples.

Curriculum Learning (CL) is the concept of introducing examples to a network in a meaningful order, most often by difficulty from 'easy' to 'hard'. The idea is that by ranking so that the network is initially exposed to simpler examples and then gradually introduced to more complex examples, the network will converge faster [1]. Though a simple concept, CL has demonstrated excellent generalisability, showing success in areas such as motif finding, noun phrase conference [7] and multi-task learning [14]. Research has also shown that self-paced CL where the ordering of examples is based on feedback from the network itself (dynamic), rather than a (static) curriculum set by a teacher [7], results in model improvements. In this way, the order in which examples are presented to the network is continuously updated, such that the curriculum presented at the start and that presented at the end of training may be vastly different. In our work we apply self-paced learning to sequence the presentation of training examples to SNNs.

3 Training and Testing with Pairs in a SNN

The SNN architecture consists of two neural networks that share identical weights and are joined at one or more layers [2]. SNNs receive pairs of examples as input to during both training and testing to develop similarity knowledge at an object-to-object level.

We first introduce the notation used in this paper to assist presentation of the different pair creation strategies. 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. $\mathcal{P}$ is a set of pairs $(p_1, ...p_n)$ that form the paired training batches for input to the SNN (see Figure 1). Each training pair, $p \in \mathcal{P}$, consists of a pair of examples, $p = (\hat{x}, x')$, where $\hat{x}$ is a pivot example whose relationship to passive example x' dictates whether the pair is of

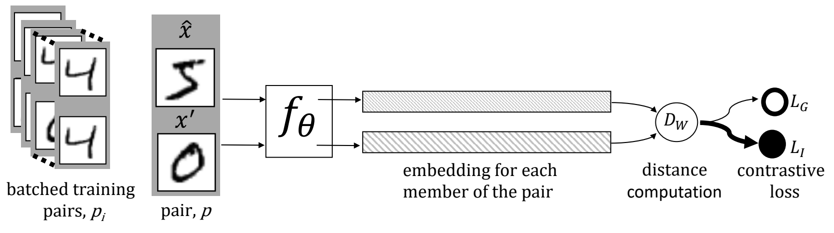

Fig. 1: The SNN learning process

class genuine, or impostor. Here the pair's relationship class is easily established by comparing class labels $y(\hat{x})$ with $y(x')$. For this we use function, $Y(p)$ or $Y(\langle \hat{x}, x' \rangle)$, which returns, p's relationship class label, such that $Y(p) = 0$ when $y(\hat{x}) = y(x')$, and $Y(p) = 1$ when $y(\hat{x}) \neq y(x')$. Typically $\hat{x}$, x' are randomly selected, to form the genuine and impostor relationship pairs for training.

During training the network develops a multi-dimensional embedding based upon the input training pairs, $\mathcal{P}$. This is facilitated by having the shared layers; essentially these layers enable the SNN to generate an embedding for each member of a pair (see Figure 1). Thereafter members can be compared using a distance metric, D_W, which influences the computation of the two loss components: loss due to pairs being further apart when they should not be, L_G; and loss due to pairs being too close when they should be further apart, L_I. Contrastive loss, L (as in Equation 3), is commonly used to guide the sub-network weight update for model learning by combining these two losses - genuine L_G and impostor L_I [3]. It essentially formulates the pair prediction error on the basis of genuine and impostor error predictions. The use of both genuine and impostor error means that the similarity metric can be directly learned by the network through the comparison of the actual pair label Y_A (equal to 0 for genuine and 1 for impostor pairs) and the distance, Euclidean or otherwise, between pair members, D_W.

This means that distance between constituents of genuine pairs are minimised over the course of training, whilst ensuring that impostor pairs maintain at least a set margin of M distance apart.

$$L_G = (1 - Y_A) \cdot {D_W}^2 \tag{1}$$

$$L_I = Y_A \cdot (max(0,\ M - D_W))^2 \tag{2}$$

$$L = L_G + L_I \tag{3}$$

The output of the identical neural networks (or 'sub-networks') form feature embeddings, f_θ, for each member of the input pair. During training it is these embeddings that are used for any distance computations, thereby ensuring iterative model refinement through contrastive loss based back propagation.

At test time, the SNN can obtain a predicted pair label, Y_P, by comparing D_W with the margin threshold M. If D_W is less than M then the network judges the pair to be genuine, otherwise it is classified as an impostor. For classification problems, a label for an unseen test example, x_q, can be obtained by pairing it with a representative training example from each class, $(\bar{x_i}, ..., \bar{x_m})$, where m is the number of classes, and $\bar{x_i}$ is a prototypical example for class i. The trained SNN is then used to provide a pair label prediction for each such pair. We then use a distance weighted voting algorithm to determine the classification of a query example from its nearest class prototype neighbour ($\bar{x_i}$). Note that only genuine pairings with $\bar{x_i}$ contribute to the classification vote:

$$vote(c_i) = \sum_{j=1}^{m} w_j * [EQ(c_i, y(\bar{x_j}))\ (1 - Y(\bar{x_j}, x_q))] \tag{4}$$

$$w_j = \frac{1}{\sqrt{(\sum_i |f_\theta^i(x_q) - f_\theta^i(\bar{x_j})|^2)}} \tag{5}$$

Here EQ compares two parameters and returns 1 when they match or 0 otherwise; in this case checks for matching class labels. The classification with the highest vote is deemed to be the classification of x_q. The weighting is based on Euclidean distance between examples using their feature embeddings from the network.

In the following sections we introduce pair creation strategies for SNN training, with strategies that are informed by knowledge about areas of difficulty (exploitation) and strategies that balance this with the need for problem space coverage (exploration).

3.1 Explorative Pair Selection

It is important to note that $\mathcal{P}$ only represents a small subset of all possible pairs which can be obtained by exhaustively pairing all examples in the training set (the total size of which would be $|\mathcal{X}|^2$). The result is that $\mathcal{P}$ gives a narrow view of instance relationships. We can improve this by initiating multiple pair selection sessions throughout training. Doing so allows us to explore the relationships between examples more thoroughly. Specifically instead of a static $\mathcal{P}$, as in Section 3, we can create a $\mathcal{P}$ for each cycle of training, where a cycle will consist of a set number of training epochs.

Algorithm 1: Algorithm to create the Explore Set

```
Explore: $\mathcal{P}_{RND}(n)$
$P_I, P_G := \emptyset$
for $i = 1 \ldots n/2$ do
    $\hat{x}$ := rnd_selection($\mathcal{X}$)
    $x'_1$ := rnd_selection($\mathcal{X}$) $\wedge\ y(\hat{x}) \neq y(x'_1)$
    $x'_2$ := rnd_selection($\mathcal{X}$) $\wedge\ y(\hat{x}) = y(x'_2)$
    $P_I := P_I \cup p(\hat{x}, x'_1)$
    $P_G := P_G \cup p(\hat{x}, x'_2)$
end
$Explore := P_I \cup P_G$
return $Explore$
```

Algorithm 1 lists the steps involved with random creation of a $Explore$ pair set, where given n, a call to $\mathcal{P}_{RND}(n)$ assigns n pairs to $\mathcal{P}$:= $\mathcal{P}_{RND}(n)$. Here $\hat{x}$ and its paired members x'_1 and x'_2 are randomly selected from $\mathcal{X}$ with the only condition that the two pairs formed must provide the necessary genuine and impostor representatives; such that $Y(P_I)$ is 0 and $Y(P_G)$ is 1.

3.2 Exploitative Pair Selection

Inspired by uncertainty sampling and boosting we can utilise information that we gain during the previous training cycle to inform pair selection for the next training cycle. Here instead of only exploring the problem space randomly, we integrate an exploitation phase such that pair selection will be guided by sampling in areas found to be 'hard' for the learner. Specifically for each $p_i \in \mathcal{P}$ we use the network's predictions, $Y(p_i)$ and associated loss to rank elements in $\mathcal{P}$. We extract the 'hardest' ranked pairs (i.e. pairs with the highest loss), from which we generate new pairs to form the exploitation set. We represent the ratio of exploit to explore as α. The main idea is to use this ratio to guide pair creation in areas of uncertainty (See Algorithm 2).

Algorithm 2: Algorithm to create the Exploit Set

```
1  Exploit: $\mathcal{P}_{NN}(\mathcal{P}')$
2  $P_I, P_G := \emptyset$
3  for $p_i \in \mathcal{P}'$ where $\mathcal{P}' \subset \mathcal{P}$ and $|\mathcal{P}'| = \alpha$ do
4      $p'_i = (NN_1(\hat{x}), NN_1(x'))$
5      if $Y(p'_i) = 0$ then
6          $P_G := P_G \cup p'_i$
7      end
8      if $Y(p'_i) = 1$ then
9          $P_I := P_I \cup p'_i$
10     end
11 end
12 $Exploit := P_I \cup P_G$
13 return $Exploit$
```

For each selected pair, we find the nearest neighbour of each member within each pair using function, NN_i. By taking the neighbours of the original difficult pair, we generalise network attention to the complex area of the space without overfitting on specific examples. These neighbours form the basis for a new pair for our training set. It is worth noting here that it is possible to develop the entire training set by using the exploit algorithm and setting α equal to 1. We found this to be detrimental to training, as the network tended to overfit to specific difficult areas, become trapped and develop a distorted feature embedding as a result. Hence we suggest using an Explore-Exploit ratio to prevent this.

3.3 Explorative and Exploitative Pair Selection

A mixed approach that allows the learner to both explore and exploit requires pair selection that can utilise pairs formed using both strategies from previous Sections 3.1 & 3.2. We accomplish this by randomly creating pairs to perform early exploration of the feature space through a 'dry run' of training the network for a small number of epochs (typically ten or less) which helps to initialise network weights. Thereafter we use the ratio α to generate a new set of exploit pairs (as in Algorithm 2); and the rest will consist of a new set of explore pairs (as in Algorithm 1).

Algorithm 3: Algorithm to combine Explore and Exploit Sets

```
1  **Explore&Exploit:** $\mathcal{P}_{HYBRID}(\mathcal{P})$
2  $b := \alpha \cdot |\mathcal{P}|$
3  for $p_i \in \mathcal{P}$ do
4      $\mathcal{L}$:= $\mathcal{L}$.append($L(\theta, p_i)$)
5  end
6  $\mathcal{P}$:= $\mathcal{P}$.sort($\mathcal{P}$, $\mathcal{L}$, <)
7  for $i = 1 \ldots b$ do
8      $\mathcal{P}$' := $\mathcal{P}$'.add($p_i$)
9  end
10 $Exploit$ := $P_{NN}(\mathcal{P}')$
11 $Explore$ := $P_{RND}(\mathcal{P} \setminus \mathcal{P}')$
12 return $Explore \cup Exploit$
```

The loss, L, for each p_i is maintained in $\mathcal{L}$, which is based on current network parameters θ. Pairs are sorted in decreasing order of loss and the top α pairs are used for exploit pair generation and the rest generated through the explore strategy. This process is repeated multiple times during training, as the areas of the feature space that the network will find complex will very likely change as the network 'learns' by refining θ. Note that NN_i's similarity computations are influenced by feature embeddings on the basis of the latest θ - i.e. they are based on activations of the last network layer at the current point in training.

It is also important to note that the suggested algorithms do not sample from a larger training set than the baseline method. It is merely the way in which instances are paired that changes. For example, in a data-budgeted scenario where only 1% of a dataset is available (such as in one of our evaluations later in this paper), these algorithms will operate within that budget and will not sample additional data from the training set.

3.4 Heuristic Ordering for Self-Paced Learning

To develop a structured ordering method for our pairs, we take inspiration from complexity measures used in neighbourhood analysis for case-based reasoning systems [12]. The basic idea is that an area is considered complex when neighbourhoods of examples are found to be non-homogeneous in terms of their class labels. We adopt this for complexity analysis C for a given pair p (instead of a single example) as in Equation 6.

$$C(p) = \frac{\sum_i \sum_j \; EQ(Y(p), Y(\langle NN_i(\hat{x}), NN_j(x')\rangle))}{\sum_i \sum_j \; (1 - EQ(Y(p), Y(\langle NN_i(\hat{x}), NN_j(x')\rangle)))} \tag{6}$$

The numerator in the complexity ratio counts the number of pairs formed in the neighbourhood that differ from the class of the original pair and denominator counts those that are of the same pair label (i.e. is it an impostor or genuine pair). Here $NN_i(.)$

denotes the ith nearest neighbour of a given example. We create all possible pairs between $\hat{x}$'s and x''s neighbourhoods. For any given pair, function Y returns the pair's class label (0 for genuine and 1 impostor). With self-paced learning we can use any of the pair selection strategies and sort pairs by the complexity metric for model training.

4 Evaluation

The aim of our experiments is two fold. Firstly, we aim to investigate the effect of incorporating a pair creation method by analysing three variations of pairing strategies: no strategy, a dynamic strategy (exploration) and an informed dynamic strategy (exploration/exploitation). Secondly, we aim to investigate the effect of complexity-based ordering, giving us an unordered and an ordered variation of each strategy and a total of six candidate approaches:

1. **BASE**: Pairs are unordered, and are not updated throughout training. As such this is a static, standard paired-training used for SNNs - the baseline (Section 3).
2. **BASE***: As BASE but now with pairs ordered (Section 3.4).
3. **DYNE**: Pairs are unordered, but pair are updated (hence dynamic) using the exploration algorithm throughout training (Section 3.1).
4. **DYNE***: As DYNE but now with pairs ordered (Section 3.4).
5. **DYNEE**: Pairs are unordered, and are updated (hence dynamic) using the explore and exploit algorithm according to some α ratio (Section 3.3).
6. **DYNEE***: As DYNEE but now with pairs ordered (Section 3.4).

We use the '*' postfix to indicate complexity-based ordering over those that have no ordering. We evaluate these algorithms using two different criteria. First, we perform a one-tail t-test to establish statistical significance at a confidence level of 95% on classification accuracy from network output on image classification, sentiment analysis and HAR tasks. Secondly, we examine each algorithm's capacity to learn over time by analysing averaged accuracy on each test set for increasing number of training epochs.

4.1 Datasets

Four datasets were used in our evaluation: MNIST, LMRD, SelfBACK-Thigh and SelfBACK-Wrist. MNIST has been used extensively for image classification [7, 6, 10] and the Large Movie Review Dataset is an extension of the popular movie review dataset [11], whilst SelfBACK is a recent dataset used for Human Activity Recognition (HAR) focused on muscular skeletal disorders [15].

Image Dataset The MNIST dataset is comprised of 70,000 images of handwritten single-digit numbers which are 28×28 pixels and have one of ten classes (the numbers zero to nine). We split this into 60,000 training and 10,000 test images. We allocated a budget of 1% of the full training set (600 images) and tested on the full test set.

Sentiment Analysis Dataset The Large Movie Review Dataset (LMRD) is comprised of 50,000 labeled film reviews scraped from the Internet Movie Database (IMDB) and evenly split between training and test sets [11]. These reviews are labeled as either 'positive' or 'negative' to create a binary classification task. Though the dataset contains a significant number of unlabeled reviews, we did not use these in our experiments.

We adopted a budget of 10% of the training set (2,500 reviews) and tested against the full test set (25,000). Reviews were preprocessed before submission to the network using Word2Vec [13] and averaging the word vectors to form a document vector [8]. This resulted in a single movie review being represented as a vector of 300 features.

HAR Datasets The SelfBACK dataset features time series data collected from 34 users performing different activities over a short period of time. Data was collected by mounting a tri-axial accelerometer on the thigh and right-hand wrist of participants at a sampling rate of 100Hz [15]. For our experiments, we remove any users that have less than 60 seconds of recorded data per activity, leaving 24 users. Data was split into 3 second windows, with 900 features per example.

Our goal in the SelfBACK dataset is to classify user activity based on minimal information pertaining to them. This is important for personalised HAR model generation to minimise demand on the user for labels [16]. Data is split into training and test sets within each user so that our training set consists of 4 windows (12 seconds) of data for each activity and the test set is the remaining data.

4.2 Experimental Setup

In MNIST and LMRD we performed our evaluation using a 10-fold cross-validation design, while in SelfBACK we divide each user into their own train and test set and average the results from all users. Each algorithm is therefore compared based upon the same initial sample as taken from the dataset, though the way in which this is exploited to form training pairs differ between algorithms.

4.3 Network Architecture

For the MNIST and LMRD datasets we used a 3-layer perceptron with a batch size of 16 for each of our sub-networks. We then trained these architectures for 100 epochs. For the SelfBACK dataset, we used a 5-layer convolutional network (as in [15]) with a batch size of 8 for each sub-network. This architecture was trained for 50 epochs to prevent overfitting on the smaller dataset. All architectures used ReLU activations and computations for test classification adopt Euclidean distance on feature embeddings at the similarity layer. Table 1 provides information on network hyperparameters.

We found that increasing regularization by decreasing batch size improves convergence speed on all methods, likely due to the limited number of examples. Decreasing the batch size gives DYNEE and DYNEE* flexibility to extract a relevant exploitation set, and offers more opportunity to update network weights appropriately.

Table 1: Summary of relevant network hyperparameters

	Sub-network (layers)	Total epochs	Exploitation ratio α
MNIST	MLP (3 Dense)	100	$^{\|P\|}/_{6}$
LMRD	MLP (3 Dense)	100	$^{\|P\|}/_{10}$
SelfBACK	Convolutional (3 Conv., 2 Dense)	50	$^{\|P\|}/_{4}$

4.4 Ratio of Exploration to Exploitation

We experimented to understand the impact of the ratio of exploration to exploitation, α, for each dataset. As identified above, over-exploiting a given dataset can cause overfitting and have negative effects on accuracy. We therefore sampled various α to determine the optimum for each dataset. We observed that high levels of exploitation are beneficial at the start of training, but cause overfitting towards the end. Intuitively, this suggests that exploitation would function optimally as a decay parameter - something we will explore in future work.

We established optimal α for each dataset: $^{|P|}/_{6}$ for MNIST, $^{|P|}/_{10}$ for LMRD and $^{|P|}/_{4}$ for both SelfBACK datasets. This means that for DynEE and DynEE*, at every training cycle, this proportion of the pairs were generated based upon exploiting knowledge from the previous training iterations and the rest of the pairs were sampled according to the explore strategy. We repeat the pair selection process every five epochs for DynE, DynE*, DynEE and DynEE*.

5 Results

Our results demonstrate that using a pairing strategy will improve network performance on all of the investigated datasets (see Table 2 and Figure 2). On every dataset using either DynE or DynEE boasts faster convergence and greater accuracy than the Base method. In one instance (MNIST), DynEE achieves a statistically significant higher optima than any other method. Though complexity-based ordering offers mixed results, we observe that an ordered method (DynE*) ultimately achieves the greatest accuracy on three of the four compared datasets. The results for each dataset appear in Table 2 with bold font used to indicate maximum accuracy for a dataset and asterisks indicating statistical significance with 0.95 confidence.

5.1 Dynamic Informed Pair Selection

On MNIST, we can distinguish that both DynEE and DynE begin to outperform Base from as little as 15 and 20 epochs respectively. The difference between DynEE and Base is statistically significant from epoch 40 onward. In LMRD we observe that DynE and DynEE reach superior performance to that of Base with only 60% and 70% of the training epochs required. Similarly, DynE and DynEE demonstrate faster convergence to optima on the SelfBACK-Wrist dataset, though not on SelfBACK-Thigh.

Table 2: Summary of algorithm performance throughout training

			Our Contributions				
	Training	BASE	BASE*	DYNE	DYNE*	DYNEE	DYNEE*
MNIST	25%	57.50	29.19	61.05	27.04	67.00	29.37
	50%	68.24	52.96	75.16	59.24	81.93*	69.09
	75%	72.03	63.47	80.76	72.78	86.78*	83.12
	100%	75.51	70.74	84.53	82.21	**90.05***	88.92
LMRD	25%	50.28	50.00	50.00	50.18	50.01	50.16
	50%	72.35	78.11	76.87	81.08*	73.08	79.52
	75%	83.66	83.56	84.36	**84.55**	84.10	84.54
	100%	83.95	83.32	84.30	84.26	84.13	84.13
SB - Wrist	20%	31.70	40.92*	31.19	28.62	33.38	31.18
	50%	58.25	64.82*	62.37	60.46	61.54	57.46
	70%	62.39	67.07	66.02	64.58	66.07	65.61
	100%	67.32	68.71	68.72	**70.49**	69.22	67.43
SB - Thigh	20%	40.79	44.46	43.12	43.76	45.31	46.03
	50%	60.91	61.24	60.78	61.30	58.11	58.24
	70%	66.31	66.84	67.59	69.37	65.71	63.71
	100%	71.81	73.89	73.46	**75.20**	74.39	70.63

On both SelfBACK datasets, the proposed methods ultimately converge to a higher optima than BASE can achieve.

These results suggest that the informed explorative and exploitative pair selection strategies present greater insight into the space than can be achieved through the static pairing method of the baseline. They support our hypothesis that a suitable training strategy can improve the performance of SNNs.

5.2 Heuristic Ordering for Self-Paced Learning

Ordering proves most effective in the LMRD dataset, where all ordered methods significantly outperform their unordered counterparts. Both DYNE* and DYNEE* reach new optima at 65% and 70% of the training epochs required for BASE to converge. In MNIST, all ordered methods performed comparably worse than their unordered counterparts. However, only BASE* underperforms the baseline and both DYNEE* and DYNE* outperform BASE from epochs 55 and 80 onwards respectively. On SelfBACK-Wrist, BASE* demonstrates statistically significant improvements over the BASE method until epoch 30 and (though DYNE* obtains the best accuracy), converges faster than other methods. On SelfBACK-Thigh, DYNE* consistently outperforms the BASE method from very early in training and ultimately achives a much superior accuracy.

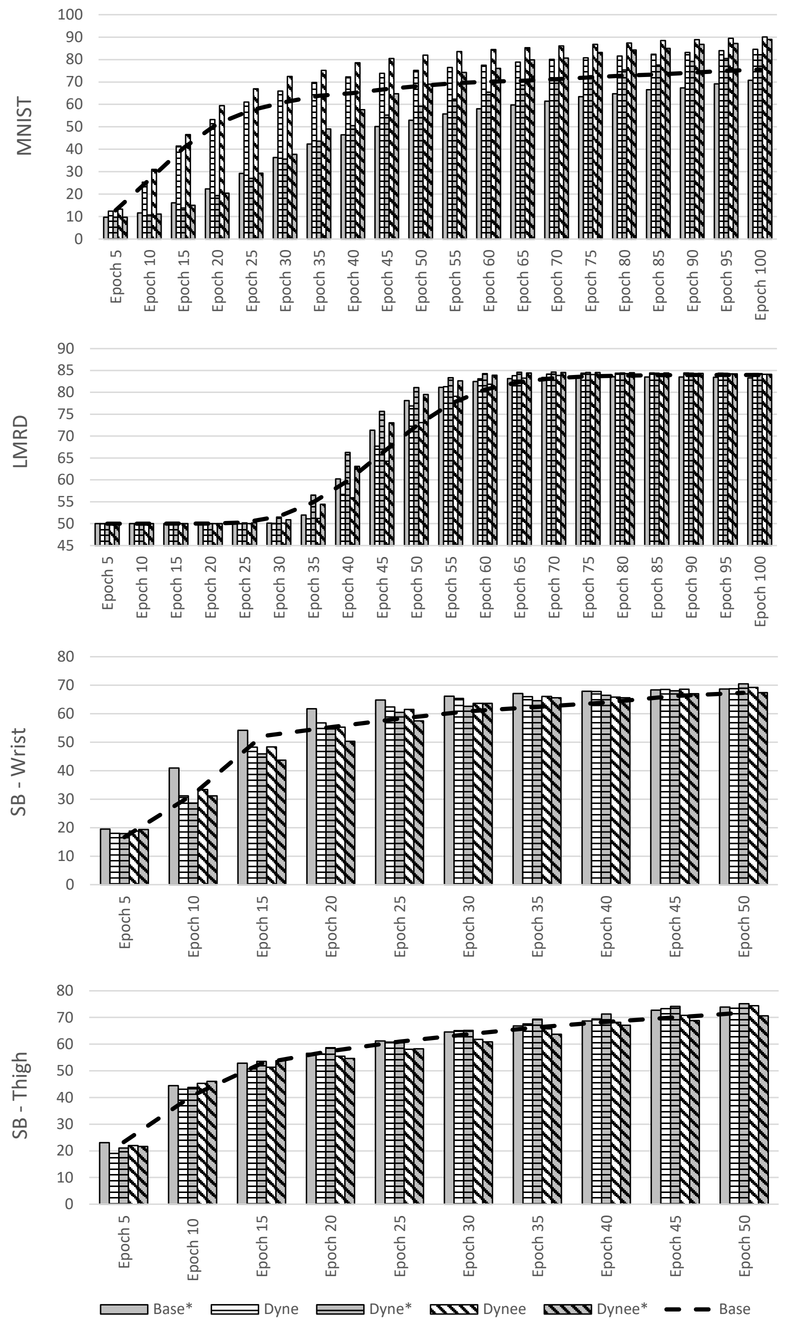

Fig. 2: Results on the MNIST, LMRD and SelfBACK datasets

We take these results as evidence that curriculum learning is effective at dealing with noisy or complex datasets. MNIST has non-complex class boundaries, suggesting that ordering will be less effective. However, both SelfBACK datasets have considerable noise, specifically the wrist dataset, as accelerometers on wrist lead to greater degrees of freedom in movement and hence is more noisy than the thigh. Additionally we observe that ordering performs well on LMRD, which is a sentiment analysis dataset. Different people can have different thresholds for enjoyment and different preferences, meaning they can describe the exact same event with different sentiments. This means there can be considerable complexity at the boundary between positive and negative classes.

In summary, the results demonstrate that a pair selection strategy will improve the performance of an SNN. In every tested dataset, the use of a non-ordered pairing strategy offers faster convergence and enables greater accuracy to be achieved, though selecting the most effective strategy depends on the task. In one domain DYNEE even allows the SNN architecture to significantly outperform the baseline in terms of classification accuracy. Furthermore, although ordering is not effective in every scenario, incorporating a sample ordering based upon neighbourhood-complexity analysis can speed up convergence on noisy or complex datasets. Specifically, DYNE* operates very effectively in these situations. This is likely because it is performing an ordered exploration of the feature space, which allows it to gradually improve its knowledge to better deal with complex areas.

6 Conclusions and Future Work

In conclusion, we have demonstrated the need for a reasoned training strategy for SNNs. We have presented four contributions towards addressing this need, highlighting the importance of a pair selection strategy and noting both an explorative algorithm, explore-exploit algorithm and an ordering method for pairs and have shown on four different datasets that each is suitable in different scenarios. Namely, informed pair selection offers a greater view on the problem space when needed and ordering allows a network to develop a structured training regime which is robust to dataset complexity. Thus we conclude that use of an appropriate pairing strategy will improve network performance, though selecting an optimal strategy is task dependant.

In future work, we aim to further optimise elements of training. We also plan to pursue a method to extend the benefits of curriculum learning further throughout training.

References

1. Bengio, Y., Louradour, J., Collobert, R., Weston, J.: Curriculum learning. In: Proc. of the 26th Annual International Conf. on Machine Learning. pp. 41 – 48. ICML '09, ACM, New York, NY, USA (June 2009)
2. Bromley, J., Guyon, I., LeCun, Y.: Signature verification using a 'siamese' time delay neural network. International Journal of Pattern Recognition and Artificial Intelligence **7**(4), 669 – 688 (August 1993)
3. Chopra, S., Hadsell, R., LeCun, Y.: 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. pp. 539 – 546. CVPR '05, IEEE Computer Society, Washington, DC, USA (June 2005)

4. Deng, K., Zheng, Y., Bourke, C., Scott, S., Masciale, J.: New algorithms for budgeted learning. Machine Learning **90**(1), 59 – 90 (January 2013)
5. Hoffer, E., Ailon, N.: Deep metric learning using triplet network. In: Feragen, A., Pelillo, M., Loog, M. (eds.) Similarity-Based Pattern Recognition. pp. 84 – 92. Springer International Publishing, Cham (2015)
6. Koch, G., Zemel, R., Salakhutdinov, R.: Siamese neural networks for one-shot image recognition. In: Deep Learning Workshop. ICML '15 (July 2015)
7. Kumar, M.P., Packer, B., Koller, D.: Self-paced learning for latent variable models. In: Advances in Neural Information Processing Systems 23, pp. 1189 – 1197. NIPS '10, Curran Associates, Inc., Red Hook, NY, USA (December 2010)
8. Le, Q., Mikolov, T.: Distributed representations of sentences and documents. In: Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32. pp. II–1188–II–1196. ICML'14, JMLR.org (2014)
9. Lizotte, D.J., Madani, O., Greiner, R.: Budgeted learning of naive-bayes classifiers. In: Proc. of the Nineteenth Conf. on Uncertainty in Artificial Intelligence. pp. 378 – 385. UAI '03, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (August 2003)
10. Loshchilov, I., Hutter, F.: Online batch selection for faster training of neural networks. In: ICLR Workshops. ICLR '16 (May 2016)
11. Maas, A.L., Daly, R.E., Pham, P.T., Huang, D., Ng, A.Y., Potts, C.: Learning word vectors for sentiment analysis. In: Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies - Volume 1. pp. 142–150. HLT '11, Association for Computational Linguistics, Stroudsburg, PA, USA (2011)
12. Massie, S., Craw, S., Wiratunga, N.: Complexity-guided case discovery for case-based reasoning. In: Proc. of the 20th AAAI Conf. on AI. pp. 216—221. AAAI Press (2005)
13. Mikolov, T., Chen, K., Corrado, G., Dean, J.: Efficient estimation of word representations in vector space. CoRR **abs/1301.3781** (2013)
14. Pentina, A., Sharmanska, V., Lampert, C.H.: Curriculum learning of multiple tasks. In: Proc. of the 2015 IEEE Computer Society Conf. on Computer Vision and Patter Recognition. pp. 5492 – 5500. CVPR '15, IEEE Computer Society, Washington, DC, USA (June 2015)
15. Sani, S., Wiratunga, N., Massie, S., Cooper, K.: Selfback - activity recognition for self-management of low back pain. In: Research and Development in Intelligent Systems XXXIII. pp. 281 – 294. SGAI '16, Springer Nature, Cham, Switzerland (December 2016)
16. Sani, S., Wiratunga, N., Massie, S., Cooper, K.: knn sampling for personalised human activity recognition. In: Proc. Intl Conf. on Case-Based Reasoning. Springer, Cham (2017)
17. Schroff, F., Kalenichenko, D., Philbin, J.: Facenet: A unified embedding for face recognition and clustering. In: Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition. pp. 815 – 823. CVPR '15, IEEE Computer Society, Washington, DC, USA (June 2015)
18. Shapire, R.E.: The boosting approach to machine learning: An overview. In: Nonlinear Estimation and Classification, pp. 149 – 172. Lecture Notes in Statistics, vol 171, Springer, New York, NY, USA (2003)
19. Vinyals, O., Blundell, C., Lillicrap, T., kavukcuoglu, k., Wierstra, D.: Matching networks for one shot learning. In: Lee, D.D., Sugiyama, M., Luxburg, U.V., Guyon, I., Garnett, R. (eds.) Advances in Neural Inf. Proc. Systems 29, pp. 3630–3638. Curran Associates, Inc. (2016)
20. Wang, J., Song, Y., Leung, T., Rosenberg, C., Wang, J., Philbin, J., Chen, B., Wu, Y.: Learning fine-grained image similarity with deep ranking. In: Proc. of the 2014 IEEE Conf. on Computer Vision and Pattern Recognition. pp. 1386 – 1393. CVPR '14, IEEE Computer Society, Washington, DC, USA (June 2014). https://doi.org/doi:10.1109/cvpr.2014.180