Learning to learn by association

Paz Ilan, 200326767

Tel Aviv University

pazilan@mail.tau.ac.il

Amit Henig, 302599069

Tel Aviv University

amithenig@mail.tau.ac.il

Abstract— *Due to the recent success of deep neural networks a need for large labeled datasets has risen. In many cases the labeled data is expensive to obtain and therefore it is desirable to be able to use unlabeled data while training to enhance performance. The following approach uses the unlabeled data by creating association cycles from the labeled data's embeddings to the unlabeled data's embeddings and back. The network utilizes the unlabeled data by adding an optimization for correct association cycles. We also add an end-to-end feature and metric learning which learns to compare embeddings with a neural network in order to be able to learn better from the unlabeled data. Our implementation can be added on top of any existing classification architecture with little effort and shows competitive results to the other approach in the subject on MNIST and better ability to generalize to more complex datasets like Fashion MNIST without additional hyper parameter tuning.*

1 INTRODUCTION

Ever since Alexnet won ILSVRC 2012, an image classification competition, a shift has occurred in the field of computer vision from traditional algorithmic solutions to neural networks based solutions. The shift was encouraged by the results in a large variety of sub subjects of the field. On the downside, due to a large amount of model parameters needed for successfully training said networks, a need for large labeled datasets has risen. Such large labeled datasets are expensive to obtain. It is therefore desirable to train machine learning models without labels (in an unsupervised fashion) or with only some fraction of the data labeled (in a semi-supervised fashion).

Recently, efforts have been made to train neural networks in an unsupervised or semi-supervised manner yielding promising results. However, most of these methods require a non-trivial architecture (GANs and encoder/decoders for example), tricks for generating data (sampling patches from an image for context prediction for example) or elaborate hand stitched metrics (left to right consistency loss for stereo depth estimation for example) for computing loss functions that don't depend on labels.

Our work aims to explore and improve a novel but intuitive approach: learning by association. This approach uses batches of labeled and unlabeled data and converts them to embeddings using a standard classification network. Then a similarity metric is used to learn closeness between embeddings and from it the probability distribution for each unlabeled embedding to be related to each of the labeled embeddings. It then uses an imaginary walker, starting from each labeled embedding and walking back and forth between the labeled and unlabeled data using the probability distribution. Thereafter it calculates the probability of each walk and tries to maximize the probability of consistent walks – the walker returning to the same class as he began. In this way the approach gains insight into improving the original networks' embeddings to generalize to the actual task. The association operation is fully differentiable and can be added to practically any existing classification network.

Our idea is to improve the above approach by adding a network to do the comparisons between embeddings rather than using a "vanilla" metric. We base our idea on the thought that not every feature carries the same weight when it comes to comparison between items, for example a car's color carries a much smaller weight in the comparison between cars than make and model. We hypothesize that by adding a network to learn feature comparisons, we can train it "on the go" to find both the comparison and the embeddings that will lead to a better use of the “walker” for semi supervised learning - i.e. make the embedding such that, along with the learned comparison, the classes are

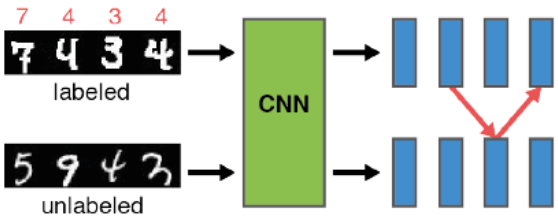
separable. We hope this will overcome the principal shortcoming of traditional metric learning based methods that the learning of the feature representation of the data and the metric are independent of each other and usually a metric is hand selected to fit the task, and therefore doesn't generalize well. We introduce new loss functions for the labeled data in order to encourage our machine to learn "basic" comparisons and other loss functions to help it learn better from the both the labeled and unlabeled data.

After adding our network along with additional optimization options we learned that our original goal was more ambitious than we initially assumed with our limited resources and experience, as the options to explore and improve become exponentially greater with each new addition we added but our resources remained limited. So instead of aiming towards improving the "vanilla" result, we aim to prove that we can get our network to similar performance as the original work while adding an enormous amount of improvement options left to explore with an easy interface to do so.

2 RELATED WORK

Various approaches for semi supervised learning have been published but some are oriented for a specific task that requires specific processing and most require non trivial extension, such as generative adversarial networks, to the network in question where as our solution, based on [1], can be attached to any classifying network.

In [1] a novel method has been introduced for semi supervised learning for classification networks. There, the machine that creates the embeddings is trained not only with the classification loss but also with a loss derived from associations calculated between the embeddings of the labeled data and of the unlabeled data. Then, an imaginary walker is sent from samples in the labeled batch to samples in the unlabeled batch and back according to a probability distribution obtained from the similarity matrix. Our work follows this method with the exception that the similarity matrix is calculated in a different manner. In [1] it is found with a pre-determined calculation, correlation between every pair of embedding vectors from the labeled and unlabeled sets. The results displayed in the paper



Figure(1): From [1], A network (green) is trained to produce embeddings (blue) that have high similarities if belonging to the same class. A differentiable association cycle (red) from embeddings of labeled (A) to unlabeled (B) data and back is used to evaluate the association

show great promise in improving classification when using a small amount of labeled data alongside more unlabeled data. We elaborate on this method further in part 3, as it is the foundation of our new approach.

In our work we attempt to train a network to find a similarity score between two embeddings in the spirit of [1]. This is conceptually similar to finding the distance between two feature vectors. In the paper [2], the authors train a shared embedding creation network along with a "decision" network to determine similarity between different patches in an image and show improvement over the state of the art in patch comparisons at the time of the article. There, as we hope in our work, the network will learn what aspects of its inputs are important when evaluating the relationship. The difference in our work is that we are motivated by grouping embeddings of the same class, which are changing as the machine trains. Also, our similarity score, which is obtained using unlabeled data, is used in order to train a network with little labeled data. Another difference is that our method will take into account similarity between more than two inputs at a time.

A method also related to helping the network learn to group is using triplet loss. The loss is used in [3] to make the network group embeddings generated from images of the same person, in comparison to embeddings generated from images of different people. This uses only labeled data for that purpose and a predefined distance metric. In one of the ways we train our similarity machine, only with labeled data, we use a similar principle where instead of comparing the similarity to the ground truth (1 if of the same class, 0 otherwise), we use a penalty derived from the distance of two similarity results,

one with an input of the same class and another with an input of a different class.

3 METHODS

We first discuss the approach in article [1], since our work is based on this approach along with our modifications and improvements.

3.1 Original approach

The approach presented in article [1] is based on the hypothesis that good embeddings will have a high similarity if they belong to the same class. Thus it aims to use both the labeled and the unlabeled data to optimize the parameters of a CNN to produce "good" embeddings - meaning embeddings that the classifier can correctly classify. This is done by passing a batch of labeled and unlabeled images (A_{img}, B_{img} respectively) through a CNN to get their embeddings (A and B). Then they imagine a walker going from A to B and back according to embedding similarities. A correct round trip is considered one where the walker ended up in the same class it started in. The goal is to maximize the probability of correct round trips between A and B. The general scheme is portrayed in figure (1).

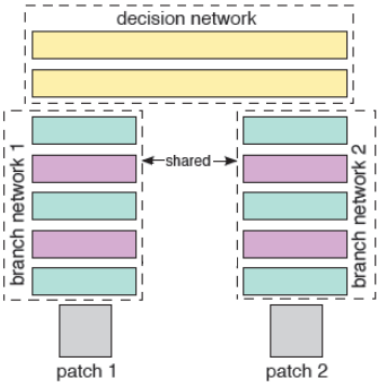
Next [1] defines the similarity between embeddings. For a batch of labeled and unlabeled embeddings a similarity matrix, denoted as M, is calculated where each element i, j is the dot product between the embedding of example i from the labeled data and the embedding of example j from the unlabeled data (the author notes that several metrics were tested and this proved to provide the best results). Mathematically:

$$(1)\ \ M_{i,j} = A_i^T \cdot B_j$$

Softmaxing the similarity matrix over columns transforms it to the transition probability matrix from A to B (each element i, j holds the probability to transfer from labeled example i to unlabeled example j):

$$(2)\ \ P_{i,j}^{AB} = P(B_j|A_i) = \left(\mathrm{softmax}_{cols}(M)\right)_{i,j} = \frac{\exp(M_{i,j})}{\sum_{j'}\exp(M_{i,j'})}$$

We can get the probabilities in the other direction by replacing M with M^T. We can now get the round trip probability:



Figure(2): (a) From [2], a basic template of a Siamese network. Inputs are first embedded independently, and then merged, usually with concatenation.

$$(3)\ \ P_{i,j}^{ABA} = \left(P^{AB}P^{BA}\right)_{i,j} = \sum_k P_{i,k}^{AB} P_{k,j}^{BA}$$

Finally, the probability of correct walks is:

$$(4)\ \ P(\text{correct walk}) = \frac{1}{|A|}\sum_{i\sim j} P_{i,j}^{ABA}$$

Where $i\sim j \Leftrightarrow \mathrm{class}(A_i) = \mathrm{class}(A_j)$.

Multiple losses were defined to achieve the stated goals:

$$(5)\ \ L_{total} = L_{walker} + L_{visit} + L_{classification}$$

Walker loss: The goal of [1]'s association cycles is not only to maximize correct walks and minimize incorrect ones, but it's also to encourage a uniform distribution of walks to the correct class, which models the idea that it is allowed and desired to end the walk on samples other than the original starting sample as long as they belong to the same class. This encouragement helps generalize the result better since starting at one sample and ending in the same one is a lot easier than getting to a different one of the same class. The walker loss is defined as the cross entropy H between the uniform target distribution of correct round trips T, calculated from the known labels, and the round trip probabilities P^{ABA}:

$$(6)\ \ L_{walker} = H(T, P^{ABA})$$

With the uniform target distribution:

$$(7)\ \ T_{i,j} = \begin{cases} \frac{1}{\#\text{of occurences of class}(A_i)\text{in A}} & i\sim j \\ 0 & \text{else} \end{cases}$$

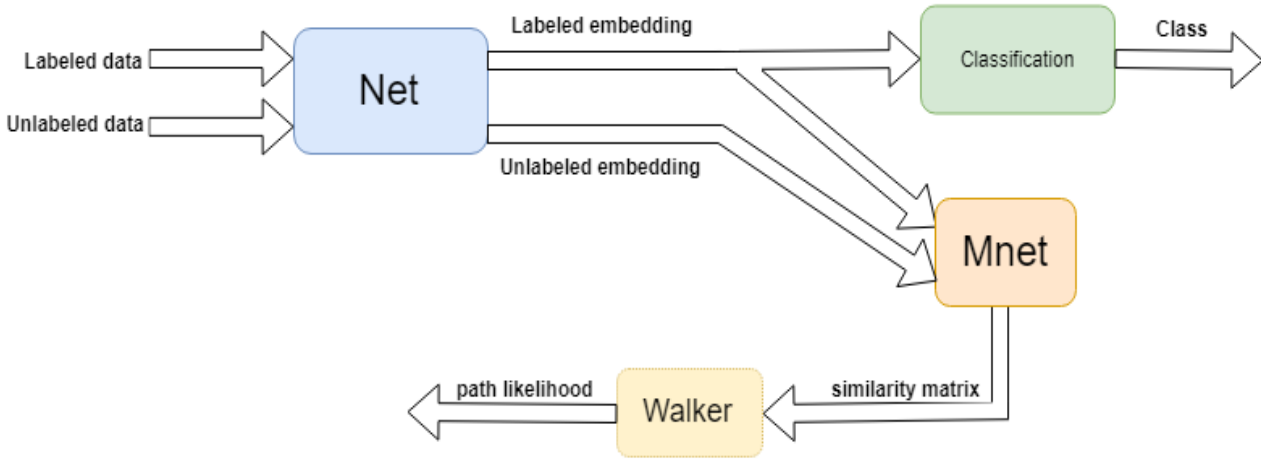


Figure (3): The general architecture of our proposed approach. We use our Mnet to calculate the similarity matrix instead of equation (1).

Visit loss: In order to enforce some unlabeled regularization and to encourage embedding to generalize better, another loss was added. This loss follows the thought that there are probably samples in the unlabeled batch that are difficult (such as badly drawn digits in MNIST) and in order to get the best use of all of the unlabeled samples, it should be beneficial to "visit" all of them, rather than just making associations among "easy" samples. The visit loss is defined as the cross entropy H between the uniform target distribution V and the visit probabilities P^{visit} (The author notes that if the unsupervised batch contains many classes that are not present in the supervised one, this regularization can be detrimental and needs to be weighted accordingly).

$$(8)\ L_{visit} = H(V, P^{visit})$$

Where the visit probability for examples in B and the uniform target distribution are defined as:

$$(9)\ P_j^{visit} = \langle P_j^{ab} \rangle_i$$

$$(10)\ V_j = \frac{1}{|B|}$$

Classification loss: So far, only the creations of embeddings and comparisons between them have been addressed. These embeddings can easily be mapped to class probabilities by adding an additional fully connected layer with softmax and a cross entropy loss on top of the network. The author calls this the classification loss, which is basically the same as the loss for a standard classification network.

3.1 Our approach

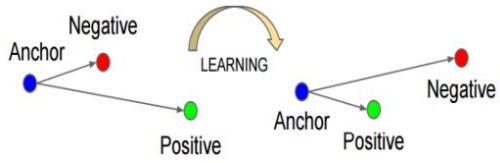
We plan to use the tools defined in [1]'s approach but the main difference is that we don't use a hand selected metric to get the similarity score between embeddings. Rather, we want a network to learn a metric alongside the embeddings. This follows the same line of thought of Siamese networks that learning a metric alongside the embedding network can improve its performance, like the one described in [2].

Most regular Siamese networks that we know of take two to three inputs at one time, pass each of them through the shared or semi-shared network and then concatenate the result (see figure (2)). We, on the other hand, want to be able to use all the embeddings, or another constant number of labeled and unlabeled batch size, at the same time to learn the similarity between every possible pair to get our own version of M from equation (1). We call this network **Mnet** (see figure (3) for the general architecture). We think our approach is the right way to solve the problem since it's more flexible, not dependent on a specific distance metric, scalable and should be able to generalize better and handle more complex data since the embeddings and comparisons are learnt jointly. This is supported by work from [3], [2] and other articles that learn comparison metric alongside the embeddings and achieve good results.

Mnet: Since we want to learn comparison between two embeddings, we wanted to use a fully connected layer and run every pair of embeddings through it. Splitting the data proved to be very slow and

inefficient on a GPU, so instead we came up with a way to run a pair-wise fully connected layer using a convolutional layer on the entire embedding matrices A,B, which runs much faster. We call this our **pair-wise fully connected convolutional** layer**,** or **PFCC** layer**,** which we will elaborate about shortly. After the **PFCC** layer, we add a cascade composed of a predetermined combination of regular fully connected layers and regular convolutional layers with 1x1xfeature_dims. The layers were chosen this way to encourage the data relevant to each comparison to be stored in a 1x1xfeature_dims cell (where feature_dims is always determined by the amount of previous layer neurons). We plan to examine different network configurations based on the above basic architecture and determine which one performs best. We assume there would be some over-fitting to our network, as there is only a small amount of labeled data, so we plan to examine the effects of adding l2 regularization and dropout to our network. We are aware that dropout is not commonly used for convolutional networks, and some argue against its use with them, but our network is a fully connected network dressed as convolutional layer for speed and convenience, so we think that dropout might help. To help with convergence and stability, we plan to use batch normalization as its shown its effectiveness in multiple cases. Since we are writing this network up from scratch, we plan on exploring a few options for the layer's activations as well, sigmoid, RELU and ELU.

Pair-wise fully connected convolutional layer: We receive our embeddings as two matrices A,B where A is NxL and B is MxL, N is the number of labeled samples, M is the number of unlabeled samples and L is the dimension of the embeddings. We begin by replicating A along its second dimension m times so that we get a new matrix A_{rep}. A_{rep} 's dimensions are NxL*M and each one of its rows holds one embedding replicated M times. We then flatten B by row stacking it so that its dimensions are 1xL*M and then replicating it N times along its first dimension so that we get a new matrix B_{rep}. B_{rep}'s dimensions are NxL*M and each of its rows holds all the embeddings of B stacked one after the other in the same order. Now, when we use a convolutional neural network with a kernel of 1xLx2 with strides of L and no padding, we get the equivalent of a fully connected layer for each pair due to the inherent weight sharing of convolutional networks.



Figure(4) from [3]: The Triplet Loss minimizes the distance between the anchor and a positive, both of which have the same identity, and maximizes the distance between the anchor and a negative of a different identity.

Since we added a new network that needs to learn comparisons from scratch, we think that only adjusting our results to use [1]'s original losses, as we described above, might be insufficient to learn the metric and embedding simultaneously, so we decided to explore a few additional loss options ideas we came up with along our experiments. The first loss is based on the desire to help Mnet learn to compare and the others are based on triplet loss. Our network architecture is closely related to Siamese networks as stated above, therefore we think that forms of triplet loss, which have proven to be successful while working with Siamese networks, could be beneficial.

Supervised comparison loss: This loss is motivated by wanting to use our labeled data in a more direct way to help our Mnet learn comparisons on top of the semi-supervised fashion described before. Mnet's output, or M_{ours}, is our equivalent of the similarity matrix defined above. This means that each element i, j in M_{ours} is meant to be a measure of how close the embeddings in location i and j from the input matrices are. We can pass them through a softmax by columns in order to transform them to transition probabilities as [1] did in his work. Following that line of thought, if we input A to our network along with itself, we will get a self similarity matrix and after the softmax we get a self transition probability matrix, or P^{AA}. We will also utilize our given labels for A to create an evenly distributed transition probability matrix, or E_{sim}. Our loss is defined as the cross entropy H between E_{sim} and our P^{AA}.Mathematically:

$$(11)\ E_{sim} = \begin{cases} \frac{1}{\#\text{of occurences of class}(A_i)\text{in A}} & i \sim j \\ 0 & \text{else} \end{cases}$$

$$(12)\ P^{AA} = \text{softmax}_{cols}(M_{ours-AA})$$

$$(13)\ L_{sc} = H\left(E_{sim}, P^{AA}\right)$$

In this fashion, we encourage our Mnet to learn comparisons and also try to help the embedding network learn to output similar embeddings for embeddings belonging to the same class. We note that since this loss gets both similarity scores between i, j and j, i separately, we hope that it will help in learning a more commutative property.

Supervised similarity triplet loss: As defined in [3], the basic triplet loss formula is:

$$(13)\ L_{triplet} = \sum_{i,j,k}\left[\left\|A_i - A_j\right\|_2^2 - \|A_i - A_k\|_2^2 + \alpha\right]_+$$

Where Ai is the embedding of image i, [i, j, k] are all image combinations such that $class(A_i) = class(A_j) \neq class(A_k)$, and $[z]_+ = \max(0, z)$.

The loss basically says that we want the difference between the closeness score (in an L2 manner) of every embedding of the same class as i and every embedding not of the same class as i, to be greater than α for all choices of embedding i. A visualization of the effect is shown in figure (4).

As we can see, the basic triplet loss formulation uses a Euclidian distance metric, which is the very thing we want to avoid (hand selecting the distance metric between embeddings). So instead, we adopt it to fit our needs. We define our similarity triplet loss, or ST loss, like so:

$$(14)\ L_{ST-supervised} = \sum_{i,j,k}\left[P_{i,k}^{AA} - P_{i,j}^{AA} + \alpha\right]_+$$

Where P^{AA}, [i, j, k] and []+ are as we defined above.

What our loss tries to enforce is that the probability that embeddings i, j (different classes) are similar will be less than the probability that embeddings i, k (same class) are similar by at least α for all possible combination. We note that from our experiments, for the ST loss, it might be more beneficial to use a sigmoid function instead of a softmax for P^{AA} to get a similarity probability rather than transition probability.

Semi-supervised similarity triplet loss: This is one adaptation we made on-top of our ST loss to fit the semi-supervised case. The difference being that we use the P^{ABA} matrix, as defined before in equation (3), and we want to increase the difference between the probability of correct walks and incorrect ones in the same fashion we described in the supervised ST loss.

$$(15)\ L_{ST-semisup} = \sum_{i,j,k}[P_{i,k}^{ABA} - P_{i,j}^{ABA} + \beta]_+$$

Where P^{ABA}, [i, j, k] and []+ are as we defined before.

Weakly supervised triplet loss: Our idea here is to try and use a triplet loss to improve P^{AB}. The problem with P^{AB} is that we don't know the classes of B at all, so we try and use a "weak" supervision form. We know that if an embedding in B is similar to several embeddings from A of the same class, it should be similar to the other from the same class as well. In this way we try to enforce the similarities we know from A to exist in B. We first try to associate each B's embedding with a specific class by averaging the similarity probabilities of the same class in A for each embedding in B, and then finding the maximum average probability. We then want the probability of similarity to all A embedding from this specific class in this B embedding to be greater than the probability of similarity to all other A embeddings for this B embedding.

$$(16)\ L_{ST-weaklysup} = \sum_{i,j,k}\left[P_{i,k}^{AB} - P_{j,k}^{AB} + \gamma\right]_+$$

Where []+ is as we defined before, P^{AB} is the same as we defined before only with a sigmoid instead of softmax to get similarity probability rather than transition probability and [i, j, k] are such that the average similarity probability of all the embedding belonging to the class of j to embedding i in B, is the maximal of all possible average class similarities for embedding i in B, and k is simply all the indices that don't meet the criteria for j for every specific i.

All of our triplet based losses have an option to do a sort of "hard mining" – instead of summing all possible differences they focus on the worst option for every example i.

Training regiment manipulation: Now, since we added an entirely different network on top of an existing one (which in itself is divided to two networks, the embedding network and the classification network) which tries to estimate similarity of changing embeddings, we thought that different combinations of when to train which variable and according to what loss, could show vastly different results. For this purpose we created an API for choosing exactly which variables train according to which loss and when (regularization

loss is always added to trained variables. Each part of the network, embeddings-Mnet-classification, has its own regularization weight).

Evaluation: In order to compare result with [1], we were forced to use the testing set as a validation set (since he did not split the training data to a validation set). Furthermore, since what we are trying to accomplish here is not simply training a specific network to specific weight values with a specific result, we need a more generalized testing scheme. So, in order to test our model in a more genuine fashion, we decided to use a different dataset for our final testing comparisons without any hyper-parameter tuning on it, including Mnet's structure, meaning that we take the model that works best for us on MNIST and [1]'s best MNIST configuration and run them both on a new dataset (we have chosen Fashion MNIST for convenience) and compare the training result there.

4 Data

4.1 MNIST

In the main portion of our experiments we use the MNIST data set [4], which is a set containing black and white 28x28 images of handwritten digits. Each sample is labeled with the digit value that's in the image, ranging from 0 to 9 for a total of 10 classes. The used MNIST data set in whole contains a training set of 60,000 examples, and a test set of 10,000 examples. In our experiments for each run, we use a total of 100 samples (10 per class) as the labeled data, and all of the images as the unlabeled data, where we don't use their labels for training the network. It is known as a good data set for people who want to try learning techniques and pattern recognition methods on real-world data while spending minimal efforts on preprocessing and formatting. Also the networks that can solve the MNIST problem aren't quite deep so a lot of experiments can be done in comparison to more complex data that requires more complex networks.

4.1 Fashion-MNIST

For final testing we've used the Fashion-MNIST data set [5], which is a set containing grayscale 28x28 images of fashion products from 10 categories. The labels are as follows: 0-T-shirt/top, 1-Trouser, 2-Pullover, 3-Dress, 4-Coat, 5-Sandal, 6-Shirt, 7-Sneaker, 8-Bag, 9-Ankle boot. The used Fashion MNIST data set in whole also contains a training set of 60,000 examples, and a test set of 10,000 examples. Fashion MNIST was intended to serve as a direct drop-in replacement for the original MNIST dataset for benchmarking machine learning algorithms. We figured this data set will be good for final testing since while it resembles MNIST in shape it has more complex data. This should help us see how well our best Mnets generalize yet still require networks that have similar training run time compared to the ones used when using MNIST.

5 Experimnets

All of the following experiments, are on MNIST, where it was significantly easier to test and develop due to its low running time and result variance in comparison to SVHN, and [1] already had a tuned hyper parameter set for it in comparison to Fashion MNIST. The total loss was minimized using Adam with the default setting. Our implementation was on top of the original implementation in Tensorflow and ran using Pycharm. After implementing all of the additions and run options mentioned in section 3, we realized we have an enormous amount of options for different networks and different hyper parameters that need to be tested. For this, we added an ability to change runtime parameters from a JSON file "on the go". This allowed us to run a large quantity of different configurations overnight from the same script.

The architecture for the embedding network for all the following experiments is:

C(32, 3) → C(32, 3) → P(2)

→ C(64, 3) → C(64, 3) → P(2)

→ C(128, 3) → C(128, 3) → P(2) → FC(128)

Here, C(n, k) stands for a convolutional layer with n kernels of size k×k and stride 1. P(k) denotes a max-pooling layer with window size k×k and stride 1. FC(n) is a fully connected layer with n output units. Convolutional and fully connected layers have exponential linear units (elu) activation functions and an additional L2 weight regularizer with weight 1e−3 applied. There is an additional FC layer, mapping the embedding to the logits for classification after the last FC layer that produces the embedding, i.e., FC(10) for 10 classes.

Unless stated otherwise, all the following runs were done with 10 labeled samples per class and the entire MNIST dataset as the unlabeled samples. To be able to compare properly the random labeled batch was chosen identically between the experiments.

First, for a base line, we present in graph (1) the results of the supervised training using only the labeled data, and the original author's network to check our performance goals. We can see [1]'s solution generalizes very well using the unlabeled data. After checking his results a few times we found that, with his suggested parameters, the variance in the final result is quite high, so in order to get a more stable, and therefore comparable, result we ran his baseline network with a learning rate of 1e-4 and no decay.

We now show the process of developing our MNet and testing Mnet's different parameters. In the displayed results, the Mnets use our process of stacking the embedding to produce an initial FC layer between each pair using a convolutional layer as described in section 3. This has reduced run time, compared to splitting the data, by a factor of 7.

First we tested several depths for structures for the network:

Shallow CONV: OurFC->C(16) → C(16)

Medium CONV: OurFC-> C(64) → C(32) → C(32) → C(16) → C(16)

Deep CONV: OurFC->C(128) → C(64) → C(64) → C(64) → C(32) → C(32) → C(32) → C(16) → C(16)

Shallow/Medium/Deep FC, are in the same structure as described above but with a fully connected layer instead of the 1x1 convolutional layer. We've tried to help Mnet cope with the changing nature of the embeddings by adding batch normalization. Also to fight over fitting from our small amount of labeled data, we've added dropout and regularization. We've seen great improvement when using normalization. Regularization also seemed to improve results. However when we've used dropout values higher than 0.05 the results gotten worse and less consistent. Different run results can be seen in graph (2). Here we couldn't have seen a clear advantage for 1x1 conv compared to a FC layer but for both, in general, we've noticed that deeper networks tends to give better results, though they have tended to be less stable for the FC options.

We then explored the effects of 3 different activation functions for our Mnet's layers- sigmoid, RELU and ELU. In our experiments we've seen a clear advantage for using the sigmoid activation. In graph (3) we display test error results for the three activations using medium conv Mnet.

Unable to match [1]'s results, we've tested the effect of the several different loss functions, as discussed in section 3, on the performance. Displayed in graph (4) are the results for deep conv Mnet with the loss combinations as described in table (1).

We've seen that the supervised comparison loss really helped our results. Also, there were some configuration (mostly the ones with only one triplet loss and without our supervised comparison loss) that performed very badly. We believe that this is due to the fact that our triplet losses are meant to support and guide the learning and not altogether dictate it, and are simply not enough to learn comparison from solely. We can see that our combined configurations mostly performed well, with a few standing out as better than the others. We can infer that our semi-supervised triplet loss only works well with our supervised triplet loss, while our weakly supervised version works well alone as well as with the supervised triplet loss. We can see that all triplet losses combined performed worse, which to us seem to indicate that we also need to play with the weights of each loss in order to manage to get the best of all worlds.

We also ran tests to check and see if using different losses on different variable training at different times could help us. The following results in graph (5) are for deep conv Mnet with a single FC layer in the middle. We note that during the training of any specific set of weights, the regularization losses for this specific set is added to the total losses. Also, a regiment that doesn't end in inf is cyclic and repeats itself in the same order until the number of epochs in the training process are over. The regiments are described in table (2). Configuration 1 basically tries to train our Mnet using all our losses for a certain amount of time, and then only train the original embedding and classifier networks using our Mnet as the comparison metric. Configuration 2 uses the same principal as one, but doesn't change embeddings weights while training the Mnet. Configuration 3 does the same as configuration one but when not training our Mnet with all the losses, it trains the embedding and Mnet using the original

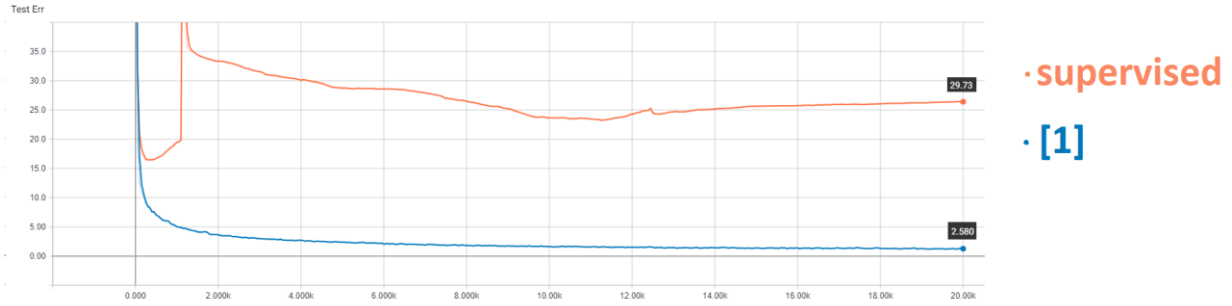
semi supervised losses with our supervised comparison loss and without any of the classifier network (loss or weights). Configuration 4 tries to train the embedding and Mnet network in a semi supervised fashion for 10k epochs before trying to train the (and with the) classifier.

There seems to be a big effect on the convergence rate by the training schedule, which is to be expected, but the end results for our well performing regimes are similar or worse than the "vanilla" regime.

Continuing to explore different approaches and playing with the hyper parameters, we've noticed that with all the new different options, deeper didn't always mean better. While converging, we got better results not from the deepest Mnet we've tried. This might be due to not having enough time to find the best of all the hyper parameters or that just that from a certain point deeper meant learning too much from the small labeled data.

Finally, we compare our most accomplished Mnet with [1]'s capability. In order to do that, we ran both options with several different seeds and in graph (6) we display the results. Our best configuration included the medium depth conv Mnet, with a single fully connected layer in the middle. It had l2 regularization weight of 1e-4, dropout of 0.01, visit loss weight of 2, logit weight of 0.1 , starting learning rate of 1e-3, decay every 500 epochs with a factor of 0.33, minimum learning rate of 5e-5 and ran with all of the available losses and variable every epoch. For [1]'s machine, we used his recommended settings.

As we can see, we achieve result which are competitive to [1]'s original results, but not better. In graph (7) we show the same configurations as for graph (6) but on Fashion MNIST. We can see that without changing any of the hyper parameters, including Mnet's architecture, chosen for MNIST specifically, we achieve a better result on a more complex dataset for every seed chosen.



Graph 1: Results for supervised training (orange) and training with [1] (blue), on MNIST with 100

Graph 2: Performance of 3 possible architectures for MNet, shallow, medium and deep for both.

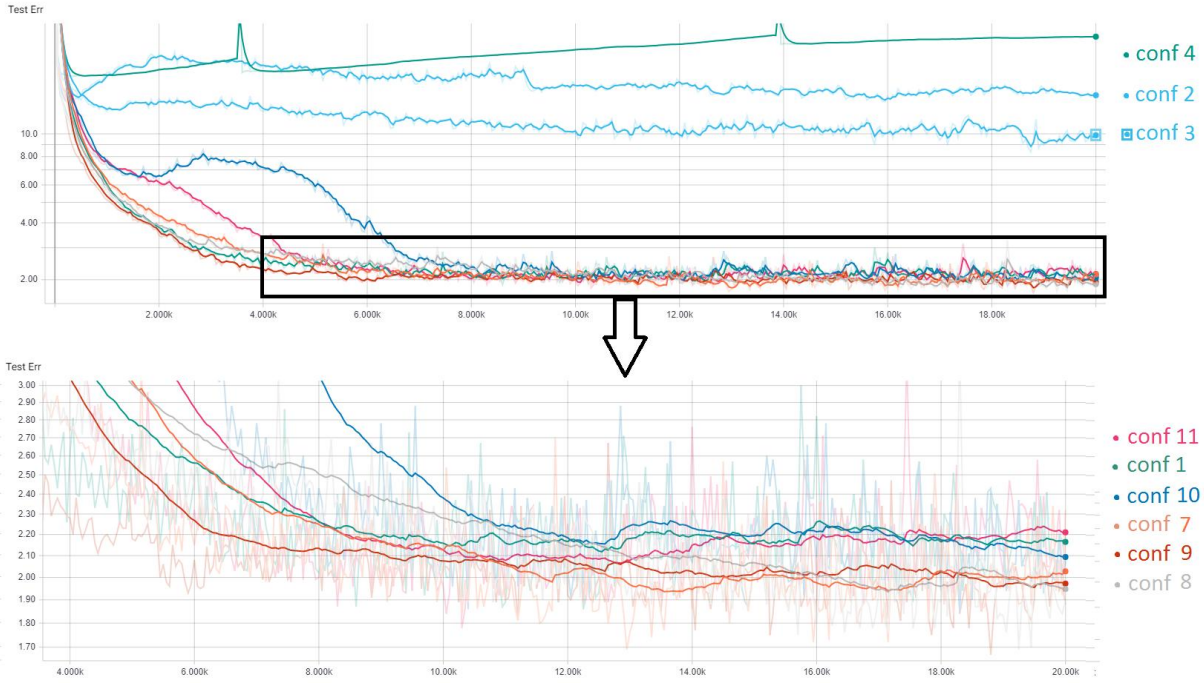
Graph 3: Sigmoid, relu and elu activation results for medium MNet.

Losses	Supervised comparison	Supervised similarity triplet	Semi-supervised similarity triple	Weakly supervised triplet
Configuration 1	*			
Configuration 2		*		
Configuration 3			*	
Configuration 4				*
Configuration 5	*	*		
Configuration 6	*		*	
Configuration 7	*			*
Configuration 8	*	*	*	
Configuration 9	*	*		*
Configuration 10	*		*	*
Configuration 11	*	*	*	*

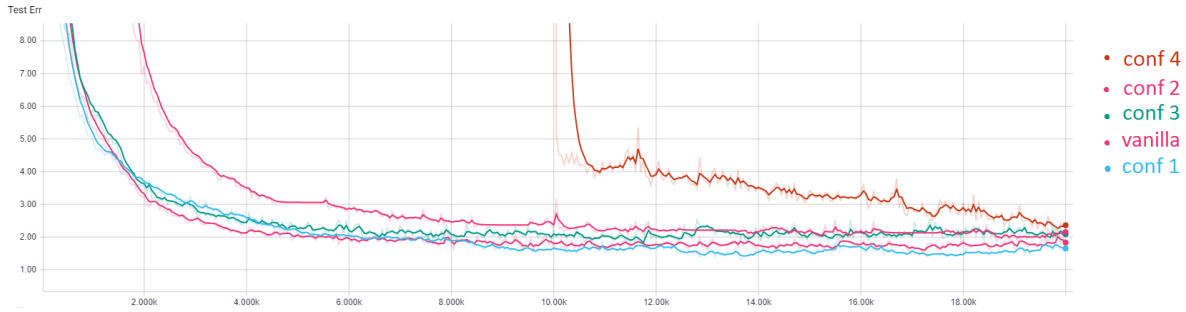
Table (1): Different loss configurations.

Configuration	Trained weights	Losses used	Number of epochs
Conf 1	Embeddings, Classifier, Mnet	Classifier, Semisup, Weakly supervised triplet, Supervised similarity triplet, Supervised comparison	1000
	Embeddings, Classifier	Classifier, Semisup	3000
Conf 2	Embeddings ,Classifier	Classifier, Semisup	500
	Mnet	Semisup, Weakly supervised triplet, Supervised similarity triplet, Supervised comparison	1000
	Embeddings, Classifier	Classifier, Semisup	2500
Conf 3	Embeddings, Classifier, Mnet	Classifier, Semisup, Supervised comparison, Weakly supervised triplet, Supervised similarity triplet	1500
	Embeddings, Mnet	Semisup, Supervised comparison	1500
Conf 4	Embeddings, Mnet	Semisup, Supervised comparison, Weakly supervised triplet	10000
	Embeddings, Classifier, Mnet	Classifier, Semisup, Supervised comparison	inf

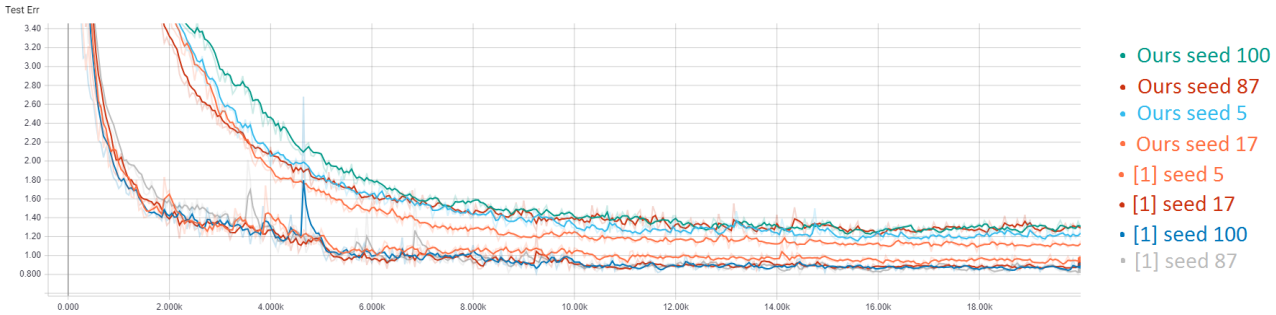
Table (2): Different training regiments.



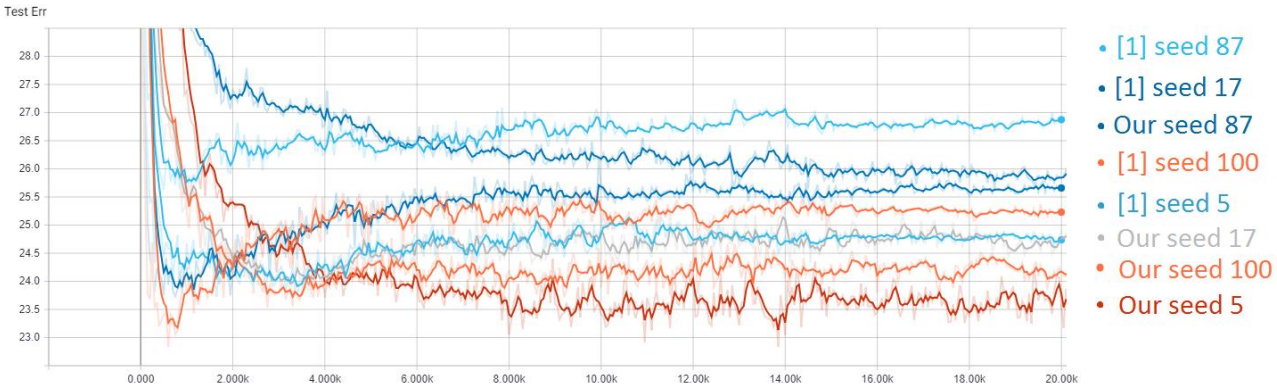
Graph 4: performance results with the configurations as described in table (1).



Graph 5: performance results with the configurations as described in table (2) and the "vanilla" regime.



Graph 6: results on MNIST of configurations tuned on MNIST, both ours and [1]s.



Graph 7: results on Fashion MNIST of configurations tuned on MNIST, both ours and [1]s.

6 Conclusion

We have managed to create an end-to-end metric and feature learning architecture that performs closely with [1]'s on a specific dataset they were both tuned on, MNIST. However, once we take our approach and [1]'s to a different dataset, Fashion MNIST, that neither of them were tuned on, we can see that we achieve superior results. From this we conclude that though our method might fall a bit short when comparing with a simple dataset that [1] specifically tuned his approach on, it has learned to generalize the training procedure better than [1] and achieve better results on the more complex dataset. It seems to us that what we see here might be a form of over fitting for the learning by association concept, that [1] did. He basically tuned his hyper parameters and judged his performance on the same dataset. So we think we managed to prove that our method can perform better in the general sense and with further tuning we believe that we can also achieve better results on MNIST.

Future extensions of our idea could be:

- Exploring the entire hyper parameter space to find the true optimal configuration for our Mnet.

- Trying to learn Mnet on more complex data for a much longer period of time.

- Adding additional losses that combine the classifier network like using the weighted average of A's classification scores with the similarity to B as the ground truth for B's classification score, or triplet loss on B's classification score to make its class more distinguishable among the classes.

- Start taking out "easy" samples from B according to their similarity results and using them as labeled data (this will enforce both the use of the "hard" samples and the additional learning from extra labeled samples).

In addition we figured out that it might be beneficial in cases where one doesn't have enough labeled data on dataset D0 with the classes he wants, but have enough labeled data of another dataset D1 with different classes then he desire, he could train the Mnet using all data D1 available and see if it helps Mnet train the network with dataset D0.

7 Appendix

Our code is available at: https://github.com/Paz87/Learning-to-learn-by-association

Original papers code is available at: https://github.com/haeusser/learning_by_association

8 References

[1] P. Haeusser, A. Mordvintsev, and D. Cremers. Learning by association - a versatile semi-supervised training method for neural networks. In IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2017

[2] Zagoruyko, S., Komodakis, N.: Learning to compare image patches via convolutional neural networks. In: CVPR 2015 (2015)

[3] F. Schroff, D. Kalenichenko, and J. Philbin. Facenet: A unified embedding for face recognition and clustering. In CVPR, 2015.

[4] Y. LeCun, C. Cortes, and C. J. Burges. The mnist database of handwritten digits, 1998. 4

[5] H. Xiao, K. Rasul, R. Vollgraf. Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms.