Probabilistic Siamese Network for Learning Representations

by

Chen Liu

A thesis submitted in conformity with the requirements
for the degree of Master of Applied Science
Graduate Department of Electrical and Computer Engineering
University of Toronto

Abstract

Probabilistic Siamese Network for Learning Representations

Chen Liu
Master of Applied Science
Graduate Department of Electrical and Computer Engineering
University of Toronto
2013

We explore the training of deep neural networks to produce vector representations using weakly labelled information in the form of binary similarity labels for pairs of training images. Previous methods such as siamese networks, IMAX and others, have used fixed cost functions such as L_1, L_2-norms and mutual information to drive the representations of similar images together and different images apart. In this work, we formulate learning as maximizing the likelihood of binary similarity labels for pairs of input images, under a parameterized probabilistic similarity model. We describe and evaluate several forms of the similarity model that account for false positives and false negatives differently. We extract representations of MNIST, AT&T ORL and COIL-100 images and use them to obtain classification results. We compare these results with state-of-the-art techniques such as deep neural networks and convolutional neural networks. We also study our method from a dimensionality reduction prospective.

Acknowledgements

I am thankful to Professor Brendan Frey whose guidance and support throughout the year made the completion of this project possible. I would also like to thank members of PSI lab, especially Hui, Andrew and Alice for their valuable help and advice.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Pattern recognition has been a long-standing problem in machine learning research. The basic pattern recognition task is to find assignments of given inputs which are useful in an application. In the context of images, a pattern recognition task can be used to find the label of an object in a given image. Various methods have been studied for classifying images. These classification methods can be categorized into supervised (trained with fully labelled data), semi-supervised (train with weakly or partially labelled data), or unsupervised (train with unlabelled data) methods. Many of these methods capture fundamental differences between patterns for classification by using hand-crafted features, which is costly and time consuming. It is important to find methods that can automatically learn quality representations from large amounts of data, so that we can achieve better classification and recognition performance. Such an improvement would be beneficial in many applications, such as medicine, security and information technology.

Under the framework of supervised learning, training a model with single image inputs is very difficult. It requires a large and fully labelled dataset to achieve good performance. In contrast, an unsupervised learning method can use unlabelled data. However, a good performance relies on sophisticated architectures with unsupervised learning, and the user has little control over the learned representations. Learning models using weak similarity labels for small collections of examples, such as pairs of input images, provides a means to discriminatively learn representations using deep architectures [1] without requiring completely labelled information [2, 3, 4, 5, 6, 7]. Furthermore, weakly labelled data can guide the algorithm to learn representations for different similarity definitions, information that cannot be easily obtained from unsupervised learning algorithms.

Figure 1.1[1] provided an example of difference in similarity definitions. If we are interested in distinguishing the *object* in the images, then the images in a) are similar because they both contain a tree. The images in b) are also similar because they both contain an airplane. The images in a) are considered to be dissimilar to the images in b) because they do not contain the same object. On the other hand, if we are interested in distinguishing the *background* of an image (scene classification), then all of the images in a) and b) are considered to be similar because they all have the same background. The underlying similarity definition allows us to tailor the representations to the precise aspect of the images that we are interested in, without the need for complex or dedicated algorithms.

In general, there are two ways to utilize similarity information. The first approach involves meth-

[1]The images were synthesized using [8] and [9].

Figure 1.1: An illustration of different semantic similarities.

ods that use similarity measurements. These methods use simple local adaptations of neighbourhood functions such as Euclidean distance or Mahalanobis distance (c.f. [10, 11, 12]). The second approach involves learning a non-linear function, such as a deep neural network that maps the input to a vector representation suitable for discriminating pairs of similar or dissimilar inputs. For example, in IMAX [2] a model is trained in order to maximize the mutual information between the pairs of representations obtained from similar input examples.

Another example of this second approach is the 'siamese architecture' [3], in which pairs of inputs are processed by two copies of the same neural network. The pair of networks are trained to make their output vectors similar for input pairs that are labelled as similar, and dissimilar for input pairs that are labelled as dissimilar. Under this setting, discriminative tasks can be performed using weakly labelled data. A more detailed description of siamese networks is given in Chapter 2.

1.1 Motivation

The discriminative task in the siamese network framework can be achieved by learning well-separated hidden representations. 'Well-separated' means that, in terms of the Euclidean distance, the hidden representations of similar input pairs should be close together and the hidden representations of dissimilar pairs should be far apart. To the best of our knowledge, prior works using the siamese network have made use of hand-chosen, fixed cost functions to achieve the desired separation in the hidden representatinos. Examples of such cost functions include mutual information [2], L_1 and L_2 norms [3, 7], and combinations using thresholds that clip off the tails of the metric distributions [6, 13].

An important issue in the above approach is that the most common cost functions are not normalized, which can bias the learned hidden representations in undesirable ways. Consequently, even in the extreme

case where the similarity labels are independent of the input pairs, certain hidden representations will still be preferred. For example, in [6] the neural network is trained to minimize the L_1 distance between hidden representations for similar pairs, and to maximize a clipped L_1 distance for dissimilar pairs. These two label-dependent costs are $C_1 = d$ and $C_0 = \max(0, \delta - d)$, where d is the L_1 norm of the difference in hidden representations and δ is a positive hand-chosen parameter. For $d = \delta$, these are $C_1 = \delta$ and $C_0 = 0$, while for $d = 2\delta$, they are $C_1 = 2\delta$ and $C_0 = 0$. Thus, if the labels are independent of the input, representations with $d = \delta$ will be preferred over representations with $d = 2\delta$. Furthermore, without normalization it is hard to consider how we should decide which of the two pairs of test images is more likely to be similar. Consider scores such as C_1, $C_1 - C_0$, $C_1/(C_1 + C_0)$, $1/(1 + e^{C_1 - C_0})$, it is not clear which one is the most appropriate to use for assessing similarity since they are all possible candidates.

1.2 Research Goals

To the best of our knowledge, no prior work exists that addresses the issues arising from hand-crafted cost functions and the lack of normalization in siamese networks. In this thesis, we formulate a probabilistic 'similarity model' in order to eliminate hand-crafted cost functions and provide normalization. The proposed similarity model takes the pairs of hidden vector representations generated by the neural networks as input and produces a normalized probability distribution over the similarity label. The proposed probabilistic model is based on maximizing the likelihood of the similarity labels for pairs of input images.

We explore several probability models and experimentally evaluate their performances. We consider models that correspond to fixed cost functions including Euclidean distance [3] and the clipped L_1 norm, as well as a number of novel models. The proposed method is compared to existing algorithms on several standard datasets and their performances are evaluated and discussed. Finally, the proposed algorithm is studied from a dimensionality reduction perspective.

1.3 Thesis Organization

In Chapter 2, we review the topics of neural networks, siamese network, neural network training, and the effects of different activation functions and momentum on neural network training. In Chapter 3, a detailed discussion of the proposed method is presented. The theoretical motivation of our work is further explained. The optimization algorithm for the proposed method is described in detail. Chapter 4 overviews the experimental setup and provides performance results for our method on several standard datasets, including AT&T face data, MNIST and COIL-100. The results are benchmarked against well-known algorithms such as Convolutional Neural Networks (CNN), Principle Component Analysis (PCA), and K-Nearest Neighbour (K-NN). Chapter 5 makes conclusions and discusses possible future directions.

Chapter 2

Background

In this chapter we introduce the fundamentals of neural networks and its extension to siamese networks. The optimization techniques used for neural network training are described, focusing on the gradient descent method. Lastly, we discuss methods for accelerating the optimization procedure such as stochastic gradient descent with mini-batch and momentum.

2.1 Neural Network

A neural network is a biologically inspired mathematical model consisting of artificial nodes called neurons or units, and interconnections that connect these neurons together to mimic a biological neural network. A neural network can be used to approximate nearly all functions. It is an adaptive system, in the sense that the interconnections can be changed by using a learning procedure.

There are many variants of neural networks, based on different combinations of three essential elements:

- Neurons
- Weighted interconnections
- Activation functions

This is illustrated by the simple one hidden-layer neural network shown in Figure 2.1.

Let the input to the neural network be the length i vector $X = \{x_1, x_2, \ldots, x_i\}$ and the output of the network be the length k vector $Y = \{y_1, y_2, \ldots, y_k\}$. Let $W^{(l)}$ represent the matrix of interconnection weights used to linearly combine the elements in the input vector to generate inputs to the l-th layer. For example, entry $w_{j,i}$ in the matrix $W^{(l)}$ is the weight between the i-th element of the input vector and the j-th unit in the l-th layer. Let $b^{(l)}$ represent the bias terms of the l-th layer. For a two-layer neural network (with one hidden layer), the output layer is denoted as $l = 2$. The hidden activity in the hidden layer $l - 1 = 1$ can be written as

$$a^{(l-1)} = (\text{hidden activity})^{(l-1)} = f(W^{(l-1)}X + b^{(l-1)}) \tag{2.1}$$

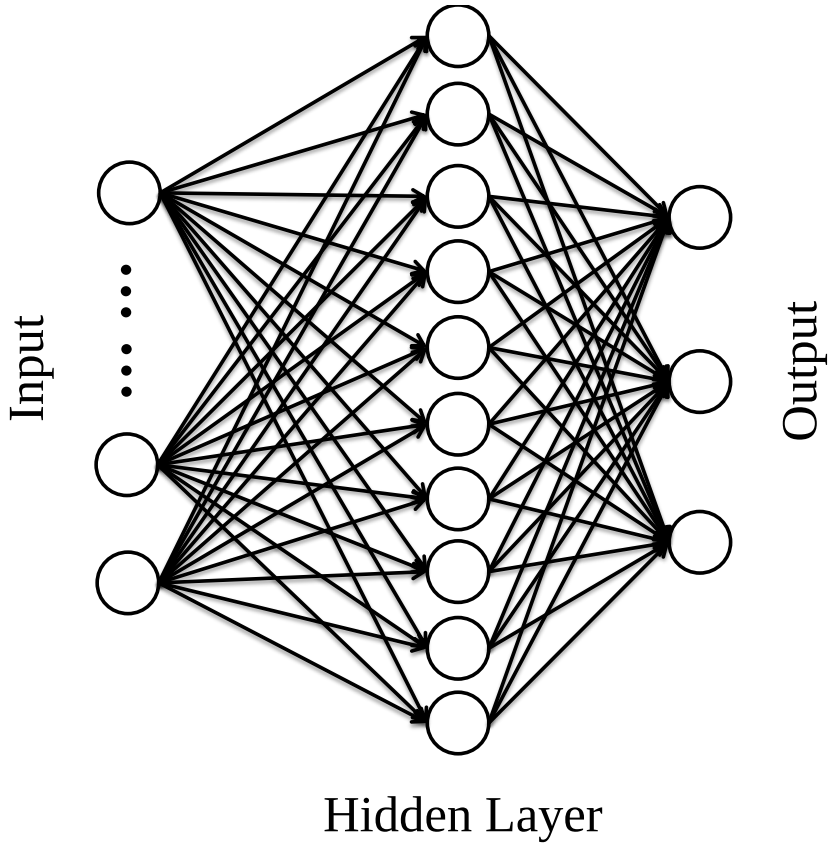


Figure 2.1: Basic Neural Network

where $f(\cdot)$ is an activation function. This activation can be propagated to the l-th layer and produce the activation for layer l, i.e. $a^{(l)}$. In our example, the activation for layer l is also the output Y given by

$$Y = a^{(l)} = f(W^{(l)} f(W^{(l-1)} X + b^{(l-1)}) + b^{(l)}) = f(W^{(l)} a^{(l-1)} + b^{(l)}). \tag{2.2}$$

In most neural network applications, an error (or cost) function is defined in terms of the inputs, outputs, and network parameters. The neural network is 'trained' by adapting its interconnection weights to minimize the error function. A solution for this optimization problem is to compute the gradient of the error function with respect to the network parameters, layer by layer, from the output layer to input layer. This algorithm is commonly referred as 'back-propagation' in neural network literature. The method of gradient descent is often used to perform network parameters updates and to find a solution to this optimization problem. We will provide details on the back-propagation algorithm in Section 2.3.

In practice, many difficult choices must be made in order to use neural networks effectively. These choices include selecting the best network architecture and the best activation function. In general, when the application is designed for a large amount of training data, more hidden units can provide better performance. When only small number of training data is available, one can still choose to use large number of hidden units, but regularization might needed to prevent overfitting. Different activation functions generally achieve similar accuracies, unless specific requirements exist on the activation for the given task. However, different activation functions can lead to very different training speeds and computation times. Table 2.1 lists several commonly used activation functions, and z is denoted as an N-dimensional input to the activation functions. The description of training speeds shown in Table 2.1 are based on common practice. The computation times (in brackets) correspond to calculating one forward pass and one derivative of a 1000×1000 random matrix on a 2.20 GHz single core CPU.

Table 2.1: Activation Functions

Name	Function	Training	Computation
Sigmoid	$\sigma = \frac{1}{1+\exp(-z)}$	Slow	Fast (0.0140)
Softmax	$S_n = \frac{\exp(z_n)}{\sum_{n=1}^{N} \exp(z_n)}$	Slow	Slow (0.2143)
Tanh	$\tanh(z) = \frac{\exp(z)-\exp(-z)}{\exp(z)+\exp(-z)}$	Medium	Medium (0.0246)
Rectified Linear Units (ReLU)	$\mathrm{ReLU} = \max(0, z)$	Fast	Fast (0.0036)

2.2 Siamese Network

A siamese network (or siamese architecture) is a particular neural network architecture consisting of two identical sub-networks with shared parameters, often used in a semi-supervised setting. An example of a siamese network is shown in Figure 2.2.

The 'siamese architecture' [3] was first proposed by using a pair of identical neural networks (with

shared parameters) that utilized similarity label information between inputs (as shown in Figure 2.2) for signature verification. The network could be trained to make the output vectors similar for input pairs that were labelled as similar and not similar for the input pairs that were labelled as dissimilar. More recently, the siamese network had been modified and applied to other tasks such as text classification [4] and speech feature classification [7]. In [5], the network was used for face verification. The authors achieved good performance by adapting the convolutional neural network architecture with an energy based model and contractive loss.

Mobahi et al. [6] proposed a similar siamese architecture for learning from video data. Pairs of input images (frames) were labelled as similar if they were close in time, and dissimilar if they were distant in time. A two-term cost function was used. The first term was a discriminative supervised function. The second term was designed to use semi-supervised similarity label information. In Chapter 1, we briefly described semi-supervised cost functions that use similarity information, and discussed several weaknesses in this type of cost functions. The semi-supervised cost function in [6] is given by

$$C_{s=1} = |h_1 - h_2| \tag{2.3}$$

$$C_{s=0} = \max(0, \delta - |h_1 - h_2|) \tag{2.4}$$

where s is a binary-valued similarity label, h_1 and h_2 are the hidden representations of a pair of inputs, and δ is a positive, hand-chosen parameter.

2.3 Optimization of the Neural Networks

2.3.1 Gradient Descent and Mini-batch Stochastic Gradient Descent

The cost functions arising from neural network optimization (such as squared error, maximum likelihood etc.) can be optimized by finding the minimum. In practice, since the dimension of the network weights can be very large and there are many local minimum, a good local minimum can achieve fairly good performance. The gradient descent (GD) algorithm is an iterative, first order optimization technique that can be applied directly to neural network training. At each iteration, gradient descent computes the gradient of the cost function $E(\cdot)$ with respect to the network parameters, and updates them according to

$$W_{t+1} = W_t - \eta \nabla E(W_t) \tag{2.5}$$

where t denotes the iteration number and $\nabla E(W_t)$ is calculated over all training examples. For a sufficiently small step size η, the algorithm is guaranteed to find a minimum on the error surface. An illustration of the gradient descent method is shown in Figure 2.3.

The gradient descent method contains several drawbacks when the amount of training data is large. First, it is expensive to compute the overall gradient at each iteration. Second, in finding the best network parameters, the optimization process tends to neglect portions of the training data, i.e. the network tends to find the best *average* parameters. The stochastic gradient descent (SGD) algorithm was developed to improve on gradient descent in the presence of a large amount of training data. In SGD, the true gradient of the error function is approximated by calculating the gradient based on a single training example X^m

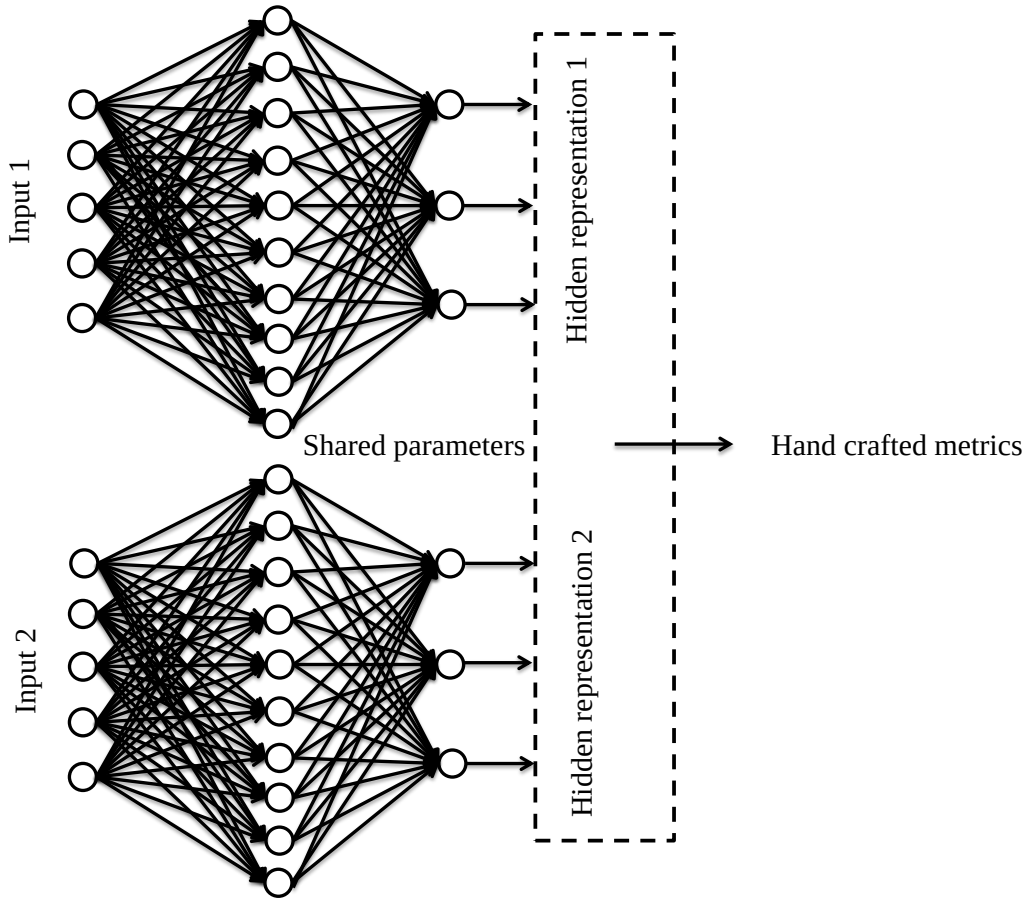


Figure 2.2: The Siamese Network with a pair of identical subnetworks, which generates hidden representations for hand-crafted metrics.

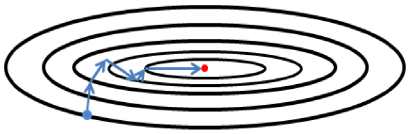


Figure 2.3: Gradient Descent Update

$$W_{t+1} = W_t - \eta \nabla E(W_t, X^m), \tag{2.6}$$

and the parameters are updated according to Algorithm 1.

Algorithm 1 Stochastic Gradient Descent

Input: *Randomly initialized W*

```
for every epoch do
    Randomly shuffle the training examples
    for m = 1, ..., M do
        Compute E(W_t, X^m)
        W_{t+1} = W_t − η∇E(W_t, X^m)
    end for
end for
```

In addition, instead of training on a single example at each iteration, the SGD algorithm can be modified to obtain its gradient by using mini-batches of training examples. Typically, a mini-batch consists of 10-100 examples [14]. In mini-batch SGD, Algorithm 1 is placed within a loop over the mini-batches, with the gradient calculated from the training examples within each mini-batch.

The benefits of mini-batch SGD are two-fold. First, it is cheaper to compute an approximate gradient for a small subset of training examples than GD on the entire dataset. First, mini-batch SGD is less noisy than SGD, as it accounts for more examples per updates. Furthermore, in every epoch, training examples are reshuffled. Therefore, different training examples are grouped into several different subsets compared to the previous epoch. The network parameters are optimized in different directions at each update, and walked multiple steps on the gradient within one epoch. It is more likely for the network to learn from the examples that are ignored in regular GD. In the end, the algorithm converges faster to a better local optimum than regular gradient descent [14].

2.3.2 Momentum

Momentum methods [15, 16] for accelerating stochastic gradient convergence has been widely studied [17, 18, 15]. In a gradient descent with momentum, the current update does not only depend on the current gradient but also depends on previous updates. When there is a long and narrow valley in the error surface, gradient descent will oscillate back and forth in the minor axis. The momentum term can set a weighted combination of previous 'velocity' and drive the update towards the minimum at a faster

rate. Other commonly used accelerating methods such as conjugate gradient and L-BFGS require a large amount of memory for computations. Therefore, momentum is a good low cost solution for speeding up the learning process in neural networks. Define $\Delta W_t = W_t - W_{t-1}$, then regular momentum is given by

$$\Delta W_t = \eta \nabla E(W_t) - \gamma \Delta W_{t-1}. \tag{2.7}$$

A variation on regular momentum called nesterov momentum was discussed in [19, 15]. If we let

$$\xi_t = W_{t-1} - \eta \nabla E(W_{t-1}), \tag{2.8}$$

then the nesterov momentum modifies regular momentum to

$$W_t = (1 + \gamma)\xi_t - \gamma \xi_{t-1} \tag{2.9}$$

where ξ_0 is initialized to zero.

We used a toy problem and conducted experiments for comparing the effect of different types of momentum, the results are shown in Table 2.2. The experiments were constructed using the squared-error function (given 1-of-N coding as the target) on MNIST data using a one hidden layer neural network (sigmoid activation functions, 250 hidden units). Twenty thousand images were sampled from the MNIST dataset for training and $10,000$ images were used for validation. The number in Table 2.2 shows the iteration when the validation accuracy reached 90%. For all three experiments, the same training and validation datasets were used, and the initialization, network architecture and cost functions were the same. By using regular momentum with gradient descent, a 3x speed up in convergence speed was achieved with sigmoid hidden units. By further modifying the momentum to nesterov momentum a 2x speed up was achieved over regular momentum on the MNIST dataset. However, the gain of using nesterov momentum is data dependent. Please refer to Appendix A for details on experiments using nesterov momentum on other datasets.

Finally, momentum scheduling [20] is yet another simple method to further improve momentum gains. An example of which is a trapezoid schedule, which consists of

1. Set an initial momentum value $0 < \gamma_0 < 1$
2. Gradually increase the momentum to a large number less than 1 (such as 0.99)
3. After few hundred epochs, gradually decrease the momentum down to γ_1

Note that it is not necessary for the momentum term to increase linearly. For example, the momentum can also be increased exponentially, or as a quadratic function over iterations.

Table 2.2: Comparisons of convergence speed using momentum on the MNIST dataset

Name	Iterations to achieve 90% Accuracy
Gradient Descent	$>$ 800 iterations
Gradient Descent + Regular Momentum	286 iterations
Gradient Descent + Nesterov Momentum	144 iterations

Chapter 3

Probabilistic Siamese Networks

In this chapter, we describe the proposed algorithm based on siamese networks. We present a number of different probabilistic similarity models and study their strengths and weaknesses. The learning procedure is discussed in detail and optimization updates are derived to allow reproduction of our results.

3.1 Similarity Model

We assume that a neural network with parameters θ is used to map an input image X to an N-dimensional hidden vector representation $\mathbf{h}$, where $\mathbf{h} = \mathbf{f}(X, \theta)$. Figure 3.1 illustrates the proposed architecture. During training, image pairs X_1 and X_2 are provided and the network is used to generate corresponding hidden vectors $\mathbf{h}_1$ and $\mathbf{h}_2$. For each input pair, a binary label s indicating whether the pair is similar ($s = 1$) or not ($s = 0$) is also provided. For instance, given the example in Figure 1.1 in Chapter 1 for object category classification, the pair of images in a) would have the similarity label $s = 1$ and the pair of images in b) would also have the similarity label $s = 1$. A pair of images with one image from a) and one image from b) would have the similarity label $s = 0$. It is our goal to train the neural networks so that the hidden vector representations for similar input pairs are close, and the hidden vector representations for dissimilar input pairs are clearly distinguishable.

The probabilistic similarity models have as input the two hidden representations $\mathbf{h}_1$ and $\mathbf{h}_2$ and outputs a distribution over the similarity label s, $P(s|\mathbf{h}_1, \mathbf{h}_2)$. Using probabilistic models allow for the normalization of the two possible predictions. We formulate the learning problem as one of maximizing the conditional likelihood of the similarity labels under a similarity model. The similarity models are parameterized by a positive vector α so that $P(s = 1|\mathbf{h}_1, \mathbf{h}_2) = g(\mathbf{h}_1, \mathbf{h}_2, \alpha)$, and $P(s = 0|\mathbf{h}_1, \mathbf{h}_2) = 1 - g(\mathbf{h}_1, \mathbf{h}_2, \alpha)$. Apart from the trivial constraint for the range of g to be in $[0, 1]$, there are several additional important properties for the similarity models:

- It should be symmetric in $\mathbf{h}_1$ and $\mathbf{h}_2$, i.e. $g(\mathbf{h}_1, \mathbf{h}_2, \alpha) = g(\mathbf{h}_2, \mathbf{h}_1, \alpha)$, since we assume that the input pairs are not given in any particular order.

- The maximum similarity should only occur when $\mathbf{h}_1 = \mathbf{h}_2$. In particular, we require that $g(\mathbf{h}_1, \mathbf{h}_2, \alpha) \leq g(\mathbf{h}_1, \mathbf{h}_1, \alpha)$ and $g(\mathbf{h}_1, \mathbf{h}_2, \alpha) \leq g(\mathbf{h}_2, \mathbf{h}_2, \alpha)$.

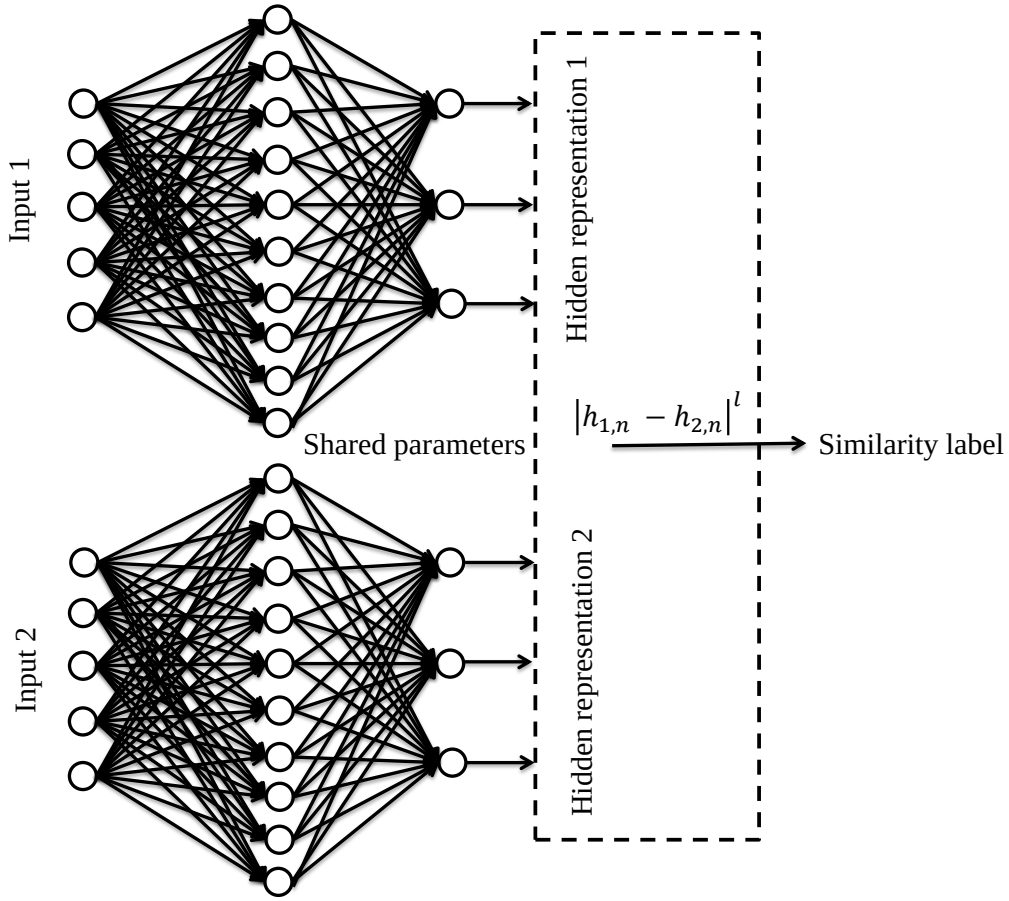


Figure 3.1: The proposed method uses a 'similarity model', which generates a probability of $P(s = 1)$.

- The similarity model should not be too flexible, or it may take over the task that the preceding neural network is meant to achieve.

We consider models that depend on weighted norms of the form $d = \sum_{n=1}^{N} \alpha_n |h_{1,n} - h_{2,n}|^\ell$, where the α's are positive coefficients that are optimized during learning. Table 3.1 lists the similarity models that we considered in this thesis. For ease of reference, we named the models by the shape of the probability $P(s = 1|\mathbf{h}_1, \mathbf{h}_2)$, i.e. $g(\mathbf{h}_1, \mathbf{h}_2, \alpha)$, as a function of $d = \sum_{n=1}^{N} \alpha_n |h_{1,n} - h_{2,n}|^\ell$. For example, the 'Gaussian' model does not, in fact, make uses of a Gaussian distribution, but the form of the dependence of P on d is that of a Gaussian density function. In the first two models, named Gaussian and flattened Gaussian, $\log P$ decreases quadratically for large d. The latter is more flat than the Gaussian model for small d. In the Laplacian model $\log P$ decreases linearly with d. The probabilistic clipped L_1 is a normalized version of the clipped L_1 cost function described previously (inspired by [6]). We will compare the representations obtained using the normalized and unnormalized versions of the L_1 cost function later in the thesis. Lastly, in the Cauchy model $\log P$ decreases as $1/d^2$, which is slower than all other models. Figure 3.2 shows plots of $\log P$ with respect to d for these models, with roughly aligned curvature when d is small. Figure 3.3 shows plots of their derivatives, which are back-propagated through the neural network during training.

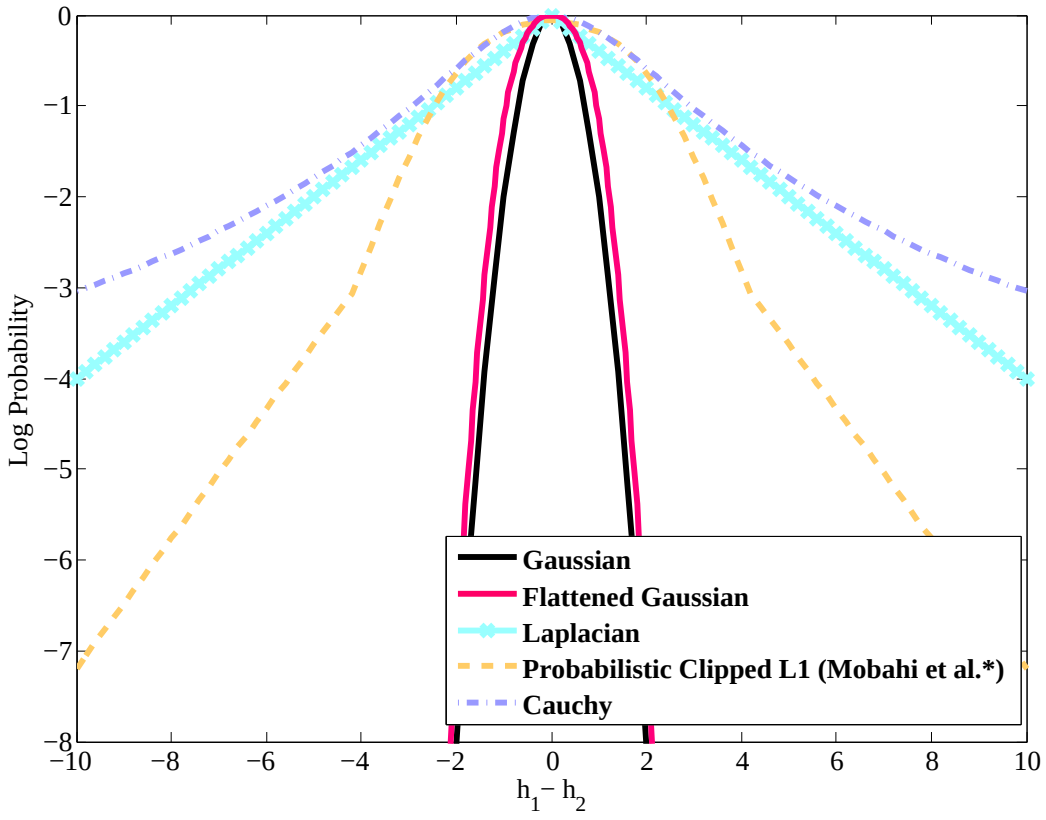


Figure 3.2: $\log P(s = 1|\mathbf{h}_1, \mathbf{h}_2)$ versus $\mathbf{h}_1 - \mathbf{h}_2$ for different similarity models.

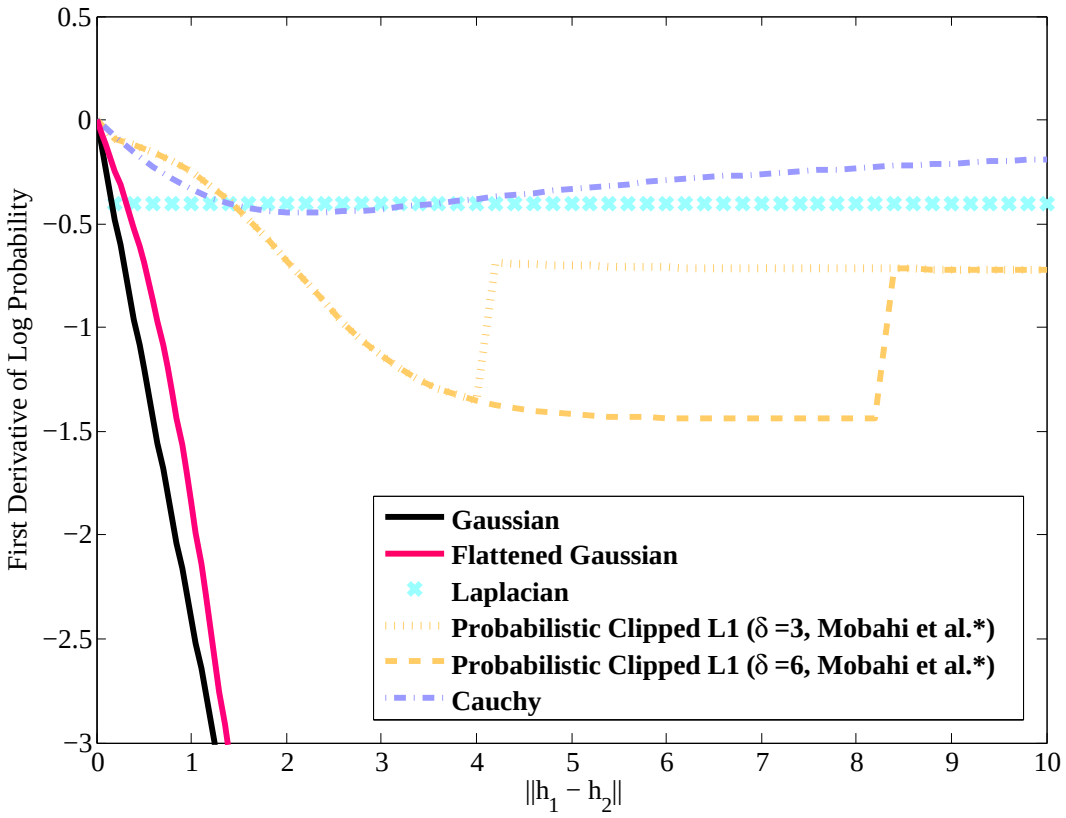


Figure 3.3: The derivative of $\log P(s = 1|\mathbf{h}_1, \mathbf{h}_2)$ w.r.t. $|\mathbf{h}_1 - \mathbf{h}_2|$, back-propagated through the neural network during training.

Table 3.1: Different forms of probabilistic similarity models.

Model	$P(s=1\|\mathbf{h}_1,\mathbf{h}_2)$
Gaussian	$\exp\left(-\sum_{n=1}^{N}\alpha_n(h_{1,n}-h_{2,n})^2\right)$
Flattened Gaussian	$(\beta+1)/\left(\beta+\exp\left(\sum_{n=1}^{N}\alpha_n(h_{1,n}-h_{2,n})^2\right)\right)$
Laplacian	$\exp\left(-\sum_{n=1}^{N}\alpha_n\lvert h_{1,n}-h_{2,n}\rvert\right)$
Probabilistic clipped L_1	$e^{-d}/(e^{-d}+e^{-\max(0,\delta-d)}),\ \ d=\sum_{n=1}^{N}\alpha_n\lvert h_{1,n}-h_{2,n}\rvert$
Cauchy	$1/(1+\sum_{n=1}^{N}\alpha_n(h_{1,n}-h_{2,n})^2)$

From Figure 3.3, we can already see that models such as the Gaussian, flattened Gaussian or probabilistic clipped L_1 likely will not perform well. If a pair of similar images are far apart in the hidden-representation space, the derivative of the Gaussian model and flattened Gaussian model will be very large, which can bias the model to fit to this particular pair. This issue is addressed by the probabilistic clipped L_1 model, but not as well as in models such as the Laplacian and Cauchy. The derivative of the Laplacian model depends on the sign of d. For all non-zero values of d, the model will penalize all data pairs equally. Thus, even though the Laplacian model can better handle outliers, it most likely will not fit to the good examples very well. Finally the Cauchy model penalizes outliers less severely than all of the other proposed model to a true outliers, and the model creates different the penalization power to typical examples (unlike in Laplacian). Therefore, we expect the Cauchy model to provide the best performance in our experiments, which are reported in Chapter 4.

3.2 Learning the Model

The training set consists of M pairs of input examples $(X_1^m, X_2^m), m = 1,\ldots,M$, and corresponding similarity labels $s^m, m = 1,\ldots,M$. In training, we maximize the conditional probability of both $s = 1$ and $s = 0$ given the hidden representations. The conditional likelihood is given by

$$P(s^1,s^2,\ldots,s^M|\mathbf{h}_1^1,\mathbf{h}_2^1,\ldots;\alpha)=\prod_{m=1}^{M}\left(g(\mathbf{h}_1^m,\mathbf{h}_2^m,\alpha)^{(s^m)}(1-g(\mathbf{h}_1^m,\mathbf{h}_2^m,\alpha))^{(1-s^m)}\right) \tag{3.1}$$

where $\mathbf{h}_1^m=\mathbf{f}(X_1^m,\theta)$ and $\mathbf{h}_2^m=\mathbf{f}(X_2^m,\theta)$. Taking the natural logarithm, we have

$$L(s^1,s^2,\ldots,s^M|\mathbf{h}_1^1,\mathbf{h}_2^1,\ldots;\alpha)=\sum_{m=1}^{M}\Big(s^m\log g(\mathbf{h}_1^m,\mathbf{h}_2^m,\alpha)+(1-s^m)\log(1-g(\mathbf{h}_1^m,\mathbf{h}_2^m,\alpha))\Big). \tag{3.2}$$

For the 2-layer neural network shown in Figure 3.1, we calculate the hidden representation for X_1^m and X_2^m by calculating the forward pass values:

$$a_1^{(1)m}=f(z_1^{(1)m})=f(W^{(1)}X_1^m+b^{(1)}) \tag{3.3}$$

$$a_2^{(1)m} = f(z_2^{(1)m}) = f(W^{(1)} X_2^m + b^{(1)}) \tag{3.4}$$

$$\mathbf{h}_1^m = a_1^{(2)m} = f(z_1^{(2)m}) = f(W^{(2)} a_1^{(1)m} + b^{(2)}) \tag{3.5}$$

$$\mathbf{h}_2^m = a_2^{(2)m} = f(z_2^{(2)m}) = f(W^{(2)} a_2^{(1)m} + b^{(2)}) \tag{3.6}$$

where $a_1^{(1)m}$ is the activation of layer 1 for the 1st image in the m-th pair of input images, $a_2^{(1)m}$ is the activation of layer 1 for the 2-nd image in the m-th pair, $b^{(1)}$ represents the bias of the input layer, and $b^{(2)}$ represents the bias of the output layer. Note that here a superscript with brackets is the index of the hidden layer, a superscript without brackets is the training case index and a subscript indicates the image within a pair of images.This notation applies to all the equations listed in Section 3.2, except equation 3.1. After calculating $\mathbf{h}_1^m$ and $\mathbf{h}_2^m$, the log-likelihood can be computed. Then the similarity model parameters α and network parameters are jointly optimized. The gradient of the log-likelihood with respect to the similarity model parameters is given by

$$\frac{\partial L}{\partial \alpha} = \sum_{m=1}^{M} \frac{s^m - g^m}{g^m(1-g^m)} \Big(\frac{\partial g(\mathbf{h}_1^m, \mathbf{h}_2^m, \alpha)}{\partial \alpha} \Big), \tag{3.7}$$

where g^m is the current vector representation for the m-th training case, $g^m = g(\mathbf{h}_1^m, \mathbf{h}_2^m, \alpha)$. The neural network is trained by back-propagating the derivatives through both $\mathbf{h}_1^m$ and $\mathbf{h}_2^m$. For example, to find the gradient with respect to the output layer weights $W^{(2)}$ we first take the derivative with respect to $\mathbf{h}_1$ and $\mathbf{h}_2$

$$\frac{\partial L}{\partial \mathbf{h}_1^m} = \sum_{m=1}^{M} \frac{s^m - g^m}{g^m(1-g^m)} \Big(\frac{\partial g(\mathbf{h}_1^m, \mathbf{h}_2^m, \alpha)}{\partial \mathbf{h}_1^m} \Big), \tag{3.8}$$

$$\frac{\partial L}{\partial \mathbf{h}_2^m} = \sum_{m=1}^{M} \frac{s^m - g^m}{g^m(1-g^m)} \Big(\frac{\partial g(\mathbf{h}_1^m, \mathbf{h}_2^m, \alpha)}{\partial \mathbf{h}_2^m} \Big). \tag{3.9}$$

Then, according to the chain rule, the derivative with respect to $W^{(2)}$ is given by

$$\frac{\partial L}{\partial W^{(2)}} = \frac{\partial L}{\partial \mathbf{h}_1^m} \frac{\partial \mathbf{h}_1^m}{\partial W^{(2)}} + \frac{\partial L}{\partial \mathbf{h}_2^m} \frac{\partial \mathbf{h}_2^m}{\partial W^{(2)}}, \tag{3.10}$$

with the last back-propagation step given by

$$\frac{\partial \mathbf{h}_1^m}{\partial W^{(2)}} = \frac{\partial \mathbf{h}_1^m}{\partial z_1^{(2)m}} \frac{\partial z_1^{(2)m}}{\partial W^{(2)}}, \tag{3.11}$$

$$\frac{\partial \mathbf{h}_2^m}{\partial W^{(2)}} = \frac{\partial \mathbf{h}_2^m}{\partial z_2^{(2)m}} \frac{\partial z_2^{(2)m}}{\partial W^{(2)}}. \tag{3.12}$$

The back-propagation expressions for the input weights $W^{(1)}$ can be obtained in a similar manner. They are

$$\frac{\partial L}{\partial W^{(1)}} = \frac{\partial L}{\partial z_1^{(1)m}} \frac{\partial z_1^{(1)m}}{\partial W^{(1)}} + \frac{\partial L}{\partial z_2^{(1)m}} \frac{\partial z_2^{(1)m}}{\partial W^{(1)}}, \tag{3.13}$$

$$\frac{\partial L}{\partial z_1^{(1)m}} = \frac{\partial L}{\partial \mathbf{h}_1^m} \frac{\partial \mathbf{h}_1^m}{\partial z_1^{(1)m}}, \tag{3.14}$$

$$\frac{\partial L}{\partial z_2^{(1)m}} = \frac{\partial L}{\partial \mathbf{h}_2^m} \frac{\partial \mathbf{h}_2^m}{\partial z_2^{(1)m}}. \tag{3.15}$$

Back-propagation can be easily generalized to l layers, where $l \in \{1, 2, \ldots, \infty\}$. The complete set of back-propagation expressions for the siamese network are given as follows:

For the output layer τ:

$$\delta_1^{(\tau)} = \frac{\partial L}{\partial \mathbf{h}_1^m} f'(z_1^{(\tau)m}) \tag{3.16}$$

$$\delta_2^{(\tau)} = \frac{\partial L}{\partial \mathbf{h}_2^m} f'(z_2^{(\tau)m}) \tag{3.17}$$

For layers $l = 1, 2, \ldots, \tau - 1$:

$$\delta_1^{(l)} = (W^{(l)} \delta_1^{(l+1)}) f'(z_1^{(l)m}) \tag{3.18}$$

$$\delta_2^{(l)} = (W^{(l)} \delta_2^{(l+1)}) f'(z_2^{(l)m}) \tag{3.19}$$

For parameter updates:

$$W^{(l)} = W^{(l)} - \eta(a_1^{(l)} \delta_1^{(l+1)} + a_2^{(l)} \delta_2^{(l+1)}) \tag{3.20}$$

$$b^{(l)} = b^{(l)} - \eta(\delta_1^{(l+1)} + \delta_2^{(l+1)}) \tag{3.21}$$

We can modify the above to equations using (2.8) and (2.9) to incorporate momentum to speed up the updates.

3.3 Algorithms

The algorithm first creates training pairs on the fly and divides them into mini-batches with an equal number of similar and dissimilar pairs. Then, for each batch we perform a forward propagation to obtain the cost. Finally, we calculate the errors for back propagation, and update the network parameters (along with the momentum according to its schedule). The proposed method is formally summarized in Algorithm 2 using the notation specified in the previous section, and the network parameters and conditional probability parameters are updated jointly.

Algorithm 2 SGD learning of neural network and similarity model

Input: *Training pairs* (X_1^m, X_2^m) *and labels* s^m, $m = 1, \ldots, M$, *architecture of the neural network and the similarity model, randomly initialized parameter values, learning rate and momentum schedule.*
Output: *Optimized network parameters, scaling parameter* α

while termination condition not satisfied **do**
  Create training pairs on-the-fly
  for *batch number* $= 1 \ldots B$ **do**
    Sample a mini-batch of K training examples
    for $k = 1, \ldots, K$ **do**
      Forward propagation:
      Compute $\mathbf{h}_1^k$, $\mathbf{h}_2^k$ and g^k
      Backward propagation:
      Compute and accumulate the derivatives w.r.t. the similarity model parameters α, i.e. $\frac{\partial L}{\partial \alpha}$ use (Eqn. 3.7)
      Compute and accumulate the derivatives w.r.t. the neural network parameters θ using back propagation, i.e. $\frac{\partial L}{\partial W^{(2)}}$(Eqn.3.10), $\frac{\partial L}{\partial W^{(1)}}$,(Eqn.3.13),...
    end for
    Update the parameter velocities using momentum and the accumulated derivatives
    Update the parameters using their velocities
  end for
end while

Chapter 4

Experiments and Analysis

In this section, we assess the performance of our proposed similarity models with siamese networks. We conducted experiments using AT&T (ORL) face data, MNIST, and COIL datasets. We examined different versions of the proposed model and directly compared with existing models such as [6]. We also evaluated the proposed model in the context of dimensionality reduction.

4.1 Dataset Preprocessing

Prior to applying machine-learning algorithms to images for training and testing purposes, data preprocessing can be performed on the images. The reasons for data preprocessing include noise reduction, dimensionality reduction, and feature extraction. Common preprocessing techniques include image whitening, Principle Component Analysis (PCA), Histogram of Oriented Gradients (HOG)[21], and Scale-Invariant Feature Transform (SIFT)[22] feature extraction. In this work, we used the PCA method to preprocess the images in a number of experiments. The PCA method was chosen because of its fast runtime, ease of implementation, and the availability of reported results for benchmarking.

PCA is an unsupervised preprocessing algorithm that uses an orthogonal transformation to convert a set of data vectors into linearly uncorrelated components referred to as the principle components. Mathematically, given a set of zero-mean input vectors $X \in \mathbb{R}^{N \times M}$ and an appropriate $K \times N$-dimensional matrix of orthogonal basis B with $K \leq N$, we have

$$\text{PCA}(X) = BX. \tag{4.1}$$

For dimensionality reduction, K is chosen to be a much smaller value than N. The key to PCA is to find the appropriate set of K orthogonal basis vectors that minimize the loss due to the linear transformation X. Formally, we perform the optimization:

$$B = \arg\min_{B} \mathbb{E}\{|x - B^T(\text{PCA}(x))|^2\} \tag{4.2}$$

where x are the columns of X.

To find the basis vectors for PCA we take the following steps:

1. Center the input data

2. Compute the covariance matrix of the input data

3. Find the eigenvalues and eigenvectors of the covariance matrix

4. Select the eigenvectors that correspond to the K largest eigenvalues as the principle components

4.2 Classification

Although the proposed model produces a single probability value, the output layer hidden-unit activities can be used for classification.

Some of the classification tasks are achieved using a L_2-Support Vector Machine (L_2 SVM) [23, 24]. In a L_2 SVM, given a set of input $\{x_1, x_2, \dots x_i, x_I\}$, and corresponding class labels $\{y_1, y_2, \dots y_i, y_I\}$, we perform classification by solving

$$\min_w \left\{ \frac{1}{2}||w||^2 + C \sum_{i=1}^{I} \max(0, 1 - y_i w x_i)^2 \right\} \tag{4.3}$$

where parameter C is chosen by validation in each experiment.

For the experiment with MNIST, a softmax classifier is also used. A softmax classifier can be considered as a one-layer neural network (no hidden layer) with a softmax activation function (Table 2.1). Given input X and class label Y, the cost function is defined as

$$E = -\frac{1}{m} \left[\sum_{m=1}^{M} \sum_{j=1}^{Class} \{Y^m = j\} \log(f(WX^m)) \right] \tag{4.4}$$

where m indexed the image, there are total of M images. W is the weight, $\{Y^m = j\}$ behaves the same as an indicator function, and $f(\cdot)$ is the softmax activation function.

In this work, we have chosen simple classification methods for the following reasons:

1. A specialized classifier for improving performance is not required

2. Data with more categories are relatively cheaper and easier to process with SVM

3. To demonstrate the ability to deepen the network for classification tasks using a softmax classifier.

4.3 Comparison of similarity models using the ORL (AT&T) face data

We used the ORL (AT&T) face dataset to evaluate our proposed similarity models. ORL face data [25] consisted of 400 gray-scale images of 40 subjects taken from 10 different views, with the eyes aligned. The original image size was 112×92 pixels, which we down-sampled to 28×28 pixels. We created a base training set by randomly selecting 5 views of each subject (200 images in total) and set aside the remaining data for testing. Figure 4.1 shows several example images, one from each subject in the dataset.

We used the mini-batch stochastic gradient descent algorithm for training. Each mini-batch was generated on-the-fly by randomly selecting 400 training pairs from the 200 base training images, so that

Figure 4.1: Examples (one of each subject) from the AT&T ORL face dataset. Images were cropped to the same size with the eyes aligned. The illuminations were normalized with uniform backgrounds.

Table 4.1: Classification performance due to different similarity models on the AT&T face data.

Model	Accuracy (Std.)
Gaussian	87.02%(0.71)
Flattened Gaussian	92.40%(1.80)
Laplacian	86.70%(2.80)
Clipped L_1 [6]	91.80%(1.88)
Probabilistic clipped L_1	89.44%(3.20)
Cauchy	94.30%(1.70)
CNN trained using class labels [6]	94.05%

an equal number of similar and dissimilar pairs are chosen. The trapezoidal momentum schedule was used with Nesterov momentum. All of the results were obtained using a $784 - 1000 - 20 - 1$ layered architecture. In the sequel, a '$784 - 1000 - 20 - 1$ layered architecture' means a neural network with an input size of 784 pixels, 1000 hidden units, 20 output units, and a binary similarity label at the last layer. For each model, training was repeated 5 times with a different random initialization of network weights and different training and test splits each time, in order to estimate the confidence intervals. For the clipped L_1 and the probabilistic clipped L_1 similarity models, we performed a search over the δ parameter using held out validation data. After training the neural network, the hidden representations from the 20 output units were fed into a L_2 SVM for classification. The best average results are reported.

Table 4.1 lists the classification accuracy under different similarity models and a state-of-the-art result obtained using a convolutional neural network [6]. The bracketed values are standard deviations of the accuracy values. In this work, we do not focus on biometric identification, thus the overall classification accuracy is inclusively reported in the table. From Table 4.1, it is shown that the Cauchy conditional probability model outperforms the other similarity models. This is expected since the Gaussian and Laplacian models do not have the appropriate tail shape, and the Cauchy model is more robust to outliers due to its tail shape. As we have argued in Chapter 3, the clipped L_1 and flattened Gaussian models do not have the appropriate first order derivatives, this is also confirmed by the experimental results. Furthermore, the Cauchy model shows a performance similar to the convolutional neural network, which is trained using the class labels themselves rather than using weak similar/dissimilar labels. This indicates that with a suitable probabilistic model it is not necessary to use a complex architecture to learn better representations. Based on the results in Table 4.1, we have decided to use the Cauchy similarity model for the remaining experiments in this chapter.

4.4 Results on MNIST handwritten digit classification

The second experiment was performed using the MNIST handwritten digits dataset. The MNIST handwritten digit dataset was collected by Dr. Yann Lecun [26], and became a popular benchmarking dataset for classification and neural network based algorithms. The MNIST handwritten digit dataset consisted of 70,000 28×28 grey-scale images of the digits 0 to 9 with a variety of writing styles. Figure 4.2 shows example images from the dataset.

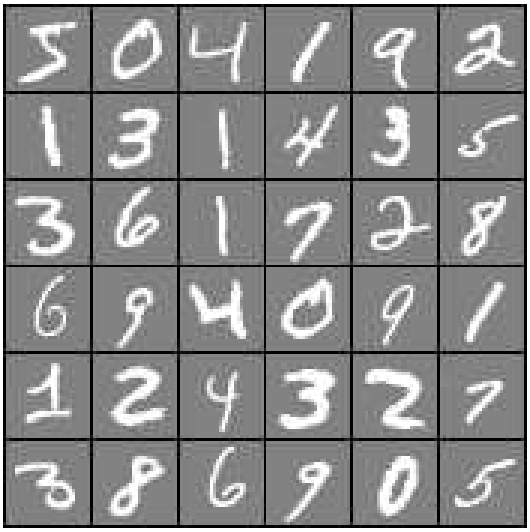


Figure 4.2: Randomly selected 36 images from the 60,000-image training set of the MNIST dataset. All of the digits are centered with a 2-pixel boundary to the boarder, on a uniform background.

Compared to the dataset used in the previous experiment, MNIST contains a smaller number of classes but has more variations within each class. It also contains a large number of training images. With the MNIST dataset, an important question to study is whether our model can perform better with more data, and if so, how much data is enough. MNIST is a relatively simple dataset. A large number of algorithms can perform well with a small number of training data, while no gains are obtained by using more training examples. In our study, we conducted experiments using different numbers of the training examples for the proposed algorithm on a network with a $784 - 60 - 30 - 1$ architecture on the original MNIST dataset. The MNIST training data consisted of 60,000 images and the test dataset consisted of 10,000 images. We randomly sampled 5000, 10000, 20000, and 50,000 training examples from the training set, and tested on the complete testing set. The resulting classification error using the hidden representations with a L_2 SVM is plotted against the number of training examples and shown in Figure 4.3. Interestingly, the algorithm performed worse than a simple classifier when there were a small number of training examples. For example, a one hidden-layer neural network classifier of 60 hidden units and no gradient acceleration could achieve around 7% error rate on MNIST. However, the algorithm showed a nearly linear performance gain with increasing training data. This is a desirable property, since our model can use all of the available training data to maximize its performance. On the other hand, the performance our model may not be optimal when only a small amount of training data available.

Other factors that can affect the performance of the proposed model are the number of hidden units and data preprocessing. In general, it is known that more hidden units and preprocessing are beneficial to representation learning. We trained the proposed algorithm with different numbers of hidden units on the original MNIST dataset, and the same dataset with preprocessing using 30 PCA components plus quadratics and 40 PCA components plus quadratics. By 'quadratics' we mean the product of every two projections on principle components. It is shown in Table 4.2 that the proposed algorithm works better

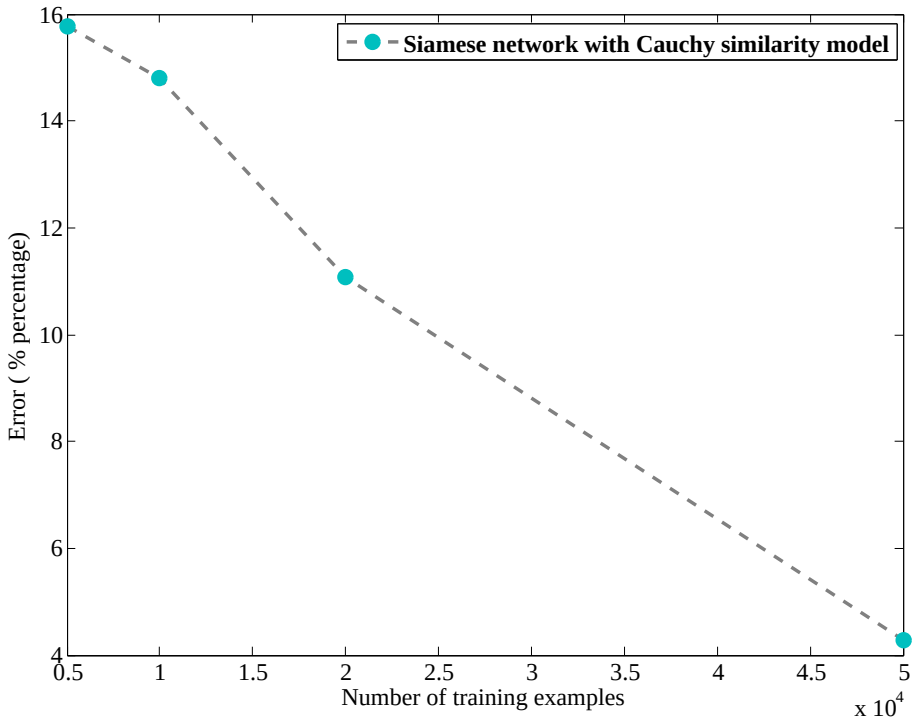


Figure 4.3: Classification accuracy with different numbers of training examples

with more hidden units and 40 PCA components plus quadratics. Note that in Table 4.2, 'Input' is 784 for images without preprocessing, 465 and 840 for '30 PCA + Quadratics' and '40 PCA + Quadratics' respectively.

Table 4.2: Classification error rates on the MNIST dataset with various numbers of hidden units and different preprocessed features.

Preprocessing	$Input - 60 - 30 - 1$	$Input - 800 - 30 - 1$	$Input - 1000 - 30 - 1$
None	4.27%	2.80%	2.60%
30 PCA + Quadratics	2.80%	2.70%	2.34%
40 PCA + Quadratics	2.68%	2.1%	1.89%

Based on the results obtained from the above experiments, in the remaining MNIST based experiments, we randomly selected $50,000$ training images as our base training set and used the remaining $10,000$ training images for validation. For training, we randomly created $50,000$ similar pairs that had the same digit, plus $50,000$ dissimilar pairs that had different digits. The network was trained by using mini-batch stochastic gradient descent with pairs taken from the $100,000$ base training pairs. Nesterov momentum was used with a trapezoid momentum schedule. We report the results on the complete set of $10,000$ testing images.

The proposed algorithm is applicable to both shallow (less than 2 hidden layers) or deep network architecture (more than 2 hidden layers). Thus, it is important to find out how many layers is considered to be enough for good classification results. The classification results of the proposed model with different hidden-layer depths are listed in Table 4.3. We considered two types of feature vectors: the original 28×28 image raster scanned into a 784-dimension vector, and features derived using 40 PCA components together with quadratic combinations, forming a feature vector of length 840 [26]. In Table 4.3, 'no hidden layer' means the network has a $784/840 - 20 - 1$ architecture, 'one hidden layer' means the network has a $784/840 - 1000 - 30 - 1$ architecture, and 'two hidden layers' means the network has a $784/840 - 1000 - 1000 - 30 - 1$ architecture without pre-training.

Table 4.3: Classification error rates on the MNIST dataset with and without preprocessing using different numbers of hidden layers.

Number of Hidden Layers	Preprocessing	Error Rate (Std.)
No hidden layer	None	$> 10\%$
No hidden layer	40 PCA + Quadratics	$> 5\%$
One hidden layer	None	2.60%(0.21)
One hidden layer	40 PCA + Quadratics	1.93%(0.14)
Two hidden layers	None	$> 5\%$
Two hidden layers	40 PCA + Quadratics	$> 5\%$

Table 4.3 shows that depending on the data and algorithm, a deep architecture is not necessarily better than a shallow architecture. In fact, for the proposed model on the MNIST dataset, a single

hidden layer architecture seems to be the most efficient.

Finally, we compare the proposed model using a single hidden layer network with several other reference techniques. The references include various shallow neural networks and the state-of-the-art for deep neural networks [27]. Once again, we consider two types of feature vectors: the original 28×28 image raster scanned into a 784-dimensional vector, and features derived using 40 PCA components, along with quadratic combinations of length 840 [26].

The classification results were obtained using a $784/840 - 1000 - 30 - 1$ architecture with sigmoid activation functions. No pre-training was used and the network weights were randomly initialized. After training the neural network using the similar/dissimilar labels, a softmax classifier applied to the pen-ultimate layer of units was trained for digit classification. Table 4.4 reports the classification results on the MNIST dataset and shows that the best results were achieved using PCA preprocessing and broad hidden layers. We ran the experiments 5 times with different random training/testing splits and different parameter initializations to obtain confidence intervals. The best error rate of our method is 1.93%, which is competitive with deep architectures that were trained using class labels and not using stochastic methods for regularization (dropout, denoising auto-encoder, etc).

Table 4.4: Classification error rates on the MNIST dataset. The first set of methods use similar/dissimilar labels to train the neural network. The second set of methods use the class labels during training.

Method	Preprocessing	Error Rate (Std.)
Clipped L_1 cost function, softmax classifier [6]	None	3.57%(0.03)
Proposed method, softmax classifier	None	2.60%(0.21)
Proposed method, softmax classifier	40 PCA + Quadratics	1.93%(0.14)
PCA+Quadratics [26]	40 PCA + Quadratics	3.30%
Deep neural network [27]	None	1.60%

4.5 Results on COIL-100 objects

We used the COIL-100 dataset for the third experiment. The dataset [28] consisted of 7200 colour images of 100 household objects with 72 different views per object. Figure 4.4 shows some images from the dataset as examples. The dataset was obtained by taking pictures of the objects for every 5 degree of rotation on a turn-table. Illumination normalization was applied to all images. In our experiment, we used gray-scale images for the experiments and the images were resized to 32×32 pixels.

We focused on two different experiments. In the first experiment, following [6], we created a training set of 400 images by taking the views with angles 0, 90, 180 and 270 degrees for each of the 100 objects. The remaining 6800 images were used for testing. Similarity was determined by object class. In the second experiment, we only labelled consecutive frames as similar. To test the effect of training set size, we formulated three sub-experiments where we randomly split the data by taking 8, 16, or 32 training images from each object category. During training, mini-batches of 100 images were generated by selecting an equal number of similar and dissimilar pairs.

Table 4.5 benchmarks the performance of the proposed method with methods trained directly using

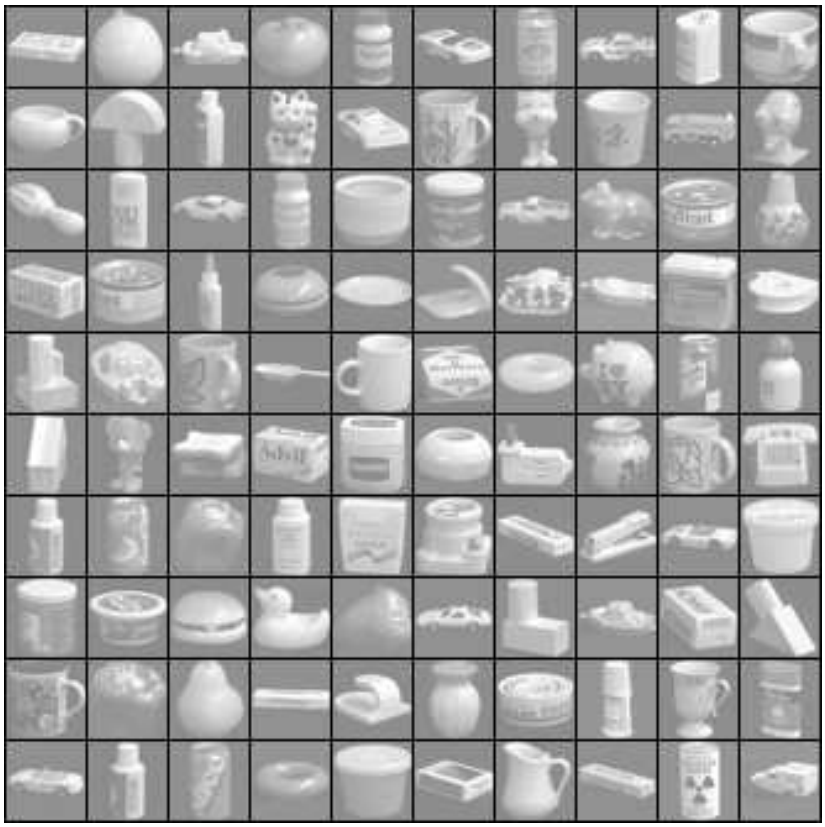


Figure 4.4: Example images (one from each category) from the COIL-100 dataset. All of the categories are centered with uniform background.

the class labels, including a linear SVM [29], a k-nearest neighbour classifier, and a convolutional neural network [6]. Our method achieved a best classification rate of 70.67%, which is comparable to the state-of-the-art under the training and testing setup that we used. Although in [29] a classification rate of 78.5% was achieved using an L_2 SVM applied to the original training images, using our more stringent test setup, the L_2 SVM achieved only 67.33%. In comparison, our method shows an improvement of approximately 3%. Compared to the convolutional neural network used in [6], we used a simpler non-convolutional architecture and achieved comparable results. Furthermore, our neural network was trained using only weak similar/dissimilar labels rather than full class labels. Our method also achieved a classification rate that is slightly better than the original clipped L_1 cost, but likely without statistical significance.

We questioned whether the first training condition, which was used in [6], was the most appropriate for this task. Thus, we experimented using the second training condition, where only consecutive frames were labelled as similar. As before, we also investigated the effect of different numbers of training cases while keeping the architecture unchanged. Table 4.6 shows the classification rates. Note that in the case

Table 4.5: Classification results on the COIL-100 dataset where images from the same class were labelled as similar. The second set of methods were trained directly using the class labels.

Method	Accuracy (Std.)
Clipped L_1 cost, L_2 SVM [6]	69.28%(0.53)
Proposed method, L_2 SVM	70.67%(0.33)
K-NN [6]	70.10%
Linear SVM [29]	78.50%
L_2 SVM	67.33%
Standard CNN [6]	71.49%

Table 4.6: Classification results on COIL-100 images with different numbers of training images and with consecutive frames labelled as similar.

Method	Numbers of training images	Accuracy (Std.)
Clipped L_1 [6](L_2 SVM)	3600	88.78%(1.35)
Proposed Method (L_2 SVM)	3600	**90.43%(0.68)**
Clipped L_1 [6](L_2 SVM)	1800	77.28%(1.10)
Proposed Method (L_2 SVM)	1800	**80.34%(1.95)**
Clipped L_1 [6](L_2 SVM)	800	68.46%(0.70)
Proposed Method (L_2 SVM)	800	**70.26%(1.20)**

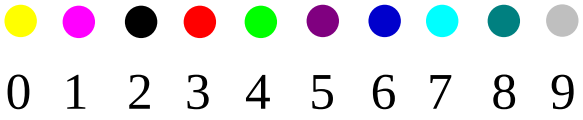
where only 800 training images were available (8 images per category), the results were averaged over 10 runs. The proposed model shows a consistently better average performance than the method described in [6].

In [6], a semi-supervised setting was introduced for temporal coherence learning in video-like images. From consistent the results shown in Table 4.5, a potential future direction of our work is an extension to video temporal coherence.

4.6 Dimensionality Reduction and Embedding

At the beginning of this chapter, we explained that one of the reasons for choosing the simple SVM and softmax classifiers for classification is to avoid relying on complex classifiers for good classification performance. To further demonstrate that the competitive results in our experiments were due to the learned hidden representations, we performed simple comparisons of 2D embeddings of the proposed model output activations (where were inputs to the simple classifiers) with those obtained under PCA and Linear Discriminant Analysis (LDA).

Figure 4.8 shows a 2-dimensional embedding of the hidden representations obtained using our method with a $784-60-2-1$ architecture. The embeddings produced by PCA and LDA are shown in Figure 4.5 and Figure 4.6, respectively. In Figure 4.6, we see that the 2D embeddings cannot be separated linearly,



(a) The colour coding of MNIST digits in Figure 4.5b, Figure 4.6, Figure 4.7, and Figure 4.8

(b) Embedding produces by PCA on the MNIST data

Figure 4.5: 2-Dimensional embedding produced by PCA on the MNIST dataset

even though they were produced using fully labelled images. PCA, as described in the previous section, does not require image labels. However, the embedding produced by using two principle components is not as good as our model. From Figure 4.8, it is clear that the proposed algorithm can learn efficient embeddings to effectively separate data using only similar/dissimilar labels. This indicates that the experimental results presented in the previous section did not rely on any specialized classifier to achieve competitive performance.

To demonstrate the benefits of a probabilistic algorithm over hand-crafted costs, we performed the experiment in [13] for dimensionality reduction using our method. In [13], the authors introduced Dimensionality Reduction by Learning an Invariant Mapping (DrLIM) with a clipped Euclidean distance similarity model. The siamese architecture used was related to [6] but did not require a supervised term in its cost function. The DrLIM algorithm consisted of a squared Euclidean distance d^2 similar model and a dissimilar model of $\max(0, \delta - d)^2$, where d is the Euclidean distance of the hidden representation

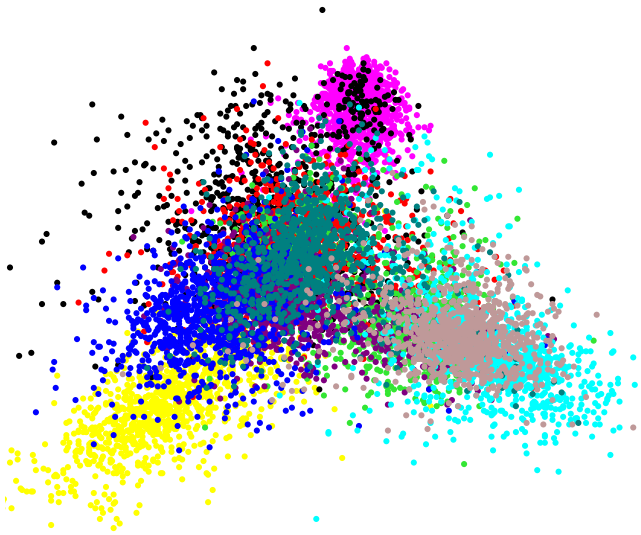


Figure 4.6: 2-Dimensional embedding produced by LDA on the MNIST dataset

of a pair of inputs and δ is a positive hand-chosen parameter. DrLIM was studied as a dimensionality reduction algorithm. In particular, experiments used the small NORB dataset [30] with one instance from the airplane class. The instance consisted of 972 96×96 pixels airplane images on uniform backgrounds. The images were taken at 18 different azimuths, 8 different elevations, and under 6 different lighting conditions. Randomly chosen example images are shown in Figure 4.9. We followed the same approach as [13] and made a randomly split of the 972 images into 660 image for training and 312 images for testing.

We performed two comparisons. First, we reproduced the experiment in DrLIM paper[13]. We provided full similarity information where similarity was defined as a pair of images taken from consecutive azimuth or elevation settings. Such similarity definition implies that the images with the same azimuth and elevation, regardless of lightings, should be mapped to the same location, i.e. lighting invariant. Following DrLIM, the 3D embeddings are presented in Figure 4.11. Second, we provided chain-like similarity information, which naturally occurs in video, where the azimuth is the fastest changing variable and lighting is the slowest changing variable. An example is shown in Figure 4.10. Along the chain, azimuth changes once per image, elevation changes once every 18 images, and lighting changes once every 9 elevations. We used identical set up, training/testing splits for both DrLIM algorithm and the proposed algorithm. The 3D embeddings are shown in Figure 4.12. Note that all of the embeddings shown in Figure 4.11 and Figure 4.12 are from the 312 test images and each colour represents an azimuth.

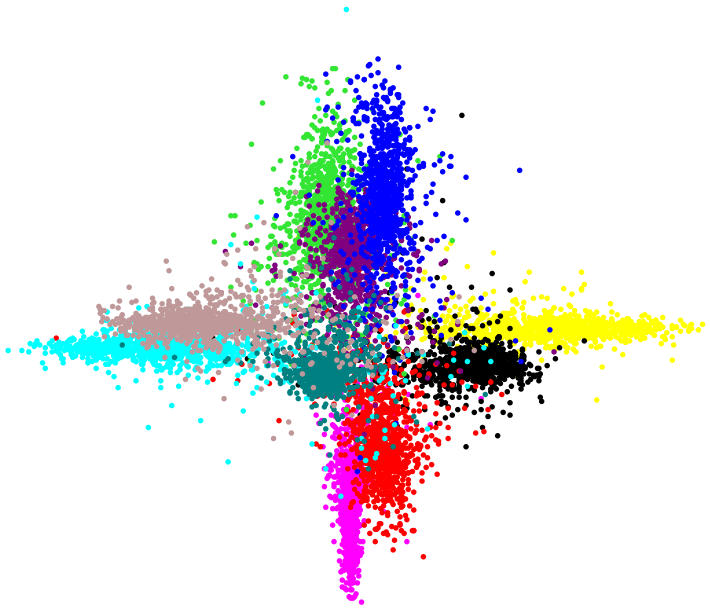


Figure 4.7: 2-Dimensional embedding produced by a neural network model uses 1-of-N labels on the MNIST dataset. The embedding is produced using a 784-60-2-10 network, and the embedding at hidden layer with 2 units is shown

We used the $9216 - 20 - 3 - 1$ architecture as indicated in the DrLIM paper[13].

When full similarity information is provided, in low dimensional space, the proposed algorithm works as well as DrLIM. The DrLIM architecture, however, is not suitable for video data with chain-like similarity information. This can be observed in Figure 4.12, where each colour represents an azimuth. Although both algorithms can distinguish between lighting conditions and map images with different lightings into small clusters as shown in Figure 4.12b and Figure 4.12d, the DrLIM algorithm does not show any easily identifiable structures in the embedding, whereas in the proposed algorithm they are easy to observe. Furthermore, the differences can be observed in Figure 4.13, where the colours represent elevations. In the chain-like similarity setup, lighting is the slowest changing variable and azimuth is the fastest changing variable. Thus, if we plot the 3D embeddings and highlight different elevations, we expect to see 9 ordered stripes of colours in one projection, organized in small clusters due to different lighting, and a rainbow of colours from another projection. This is observed in Figure 4.13c and Figure 4.13d, but not clearly observed in the equivalent DrLIM experiments.

This improvement is likely due to the normalized probability and heavy-tailed Cauchy-like distribution in our model. Under the full similarity setting, images with a small distance in pixel space are considered to be similar (neighbours). This is a natural and simple constraint that can be imposed on many dimensionality reduction algorithms for finding embeddings . However, under the chain-like similarity setting, similarity is not closely related to the distance between pairs of images in pixel space. For example, in Figure 4.10, the first image in the bottom row (start of the example chain) and the first

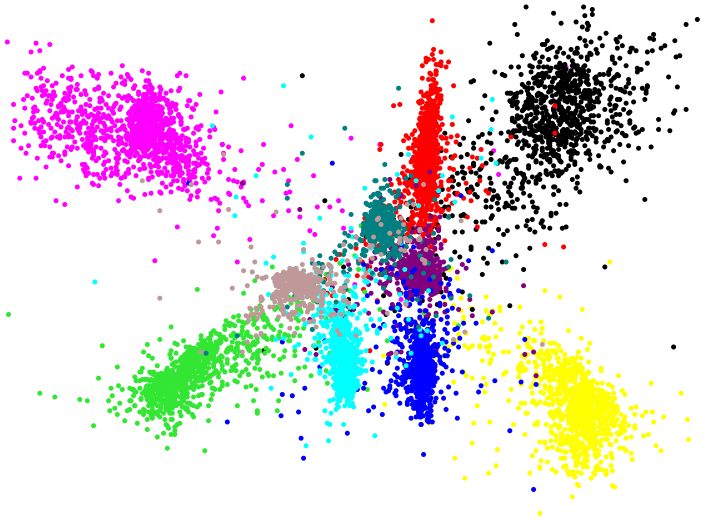


Figure 4.8: 2-Dimensional embedding produced by proposed model on the MNIST dataset. The embedding is produced using a $784 - 60 - 2 - 1$ network, and the embedding at output layer with 2 output units is shown

image in the second row from the top are close in pixel space. However, under the chain-like similarity setting, they are considered to be dissimilar. The learning task is difficult in this case, because the algorithm have to either extract the underlying representations well or be robust to outliers. Evidently, the DrLIM cost functions do not perform well in either case. Thus, from a dimensionality reduction perspective, our experiments show that the proposed probabilistic model performs better than hand-crafted models.

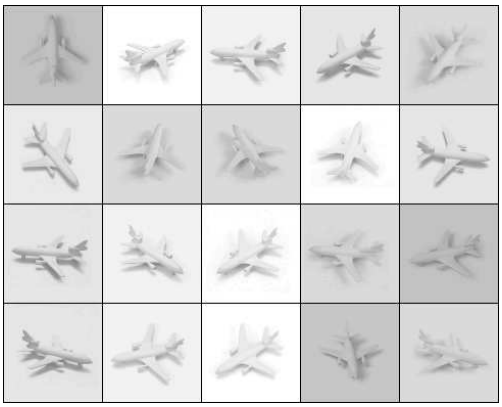


Figure 4.9: Image examples from the NORB (small) dataset

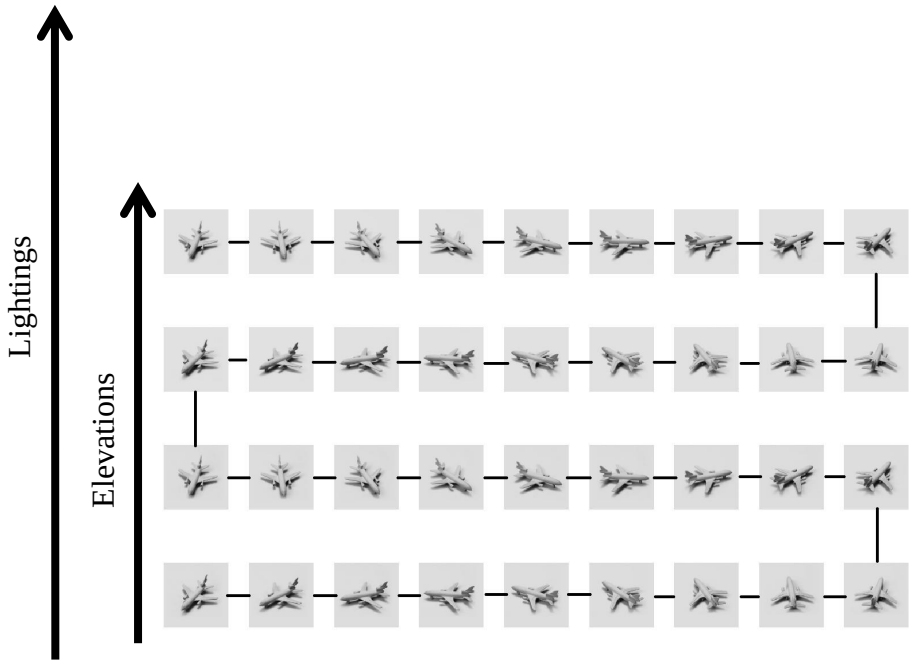
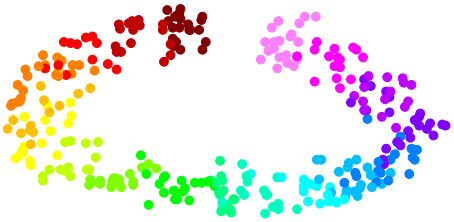
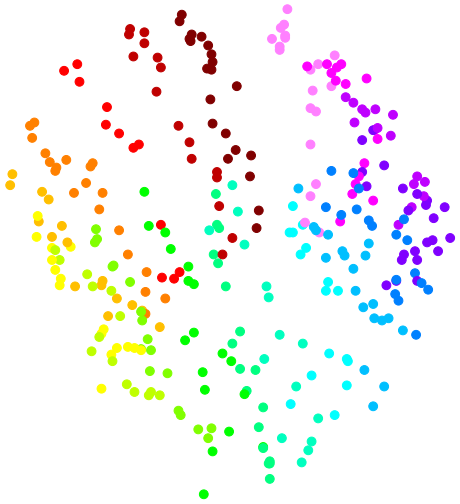


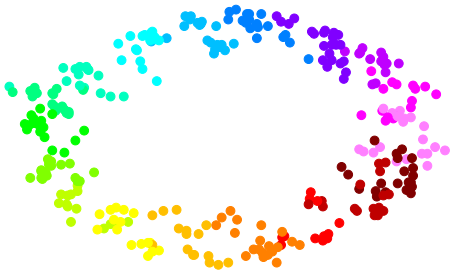
Figure 4.10: An example of chain-like similarity used in the experiment with NORB data



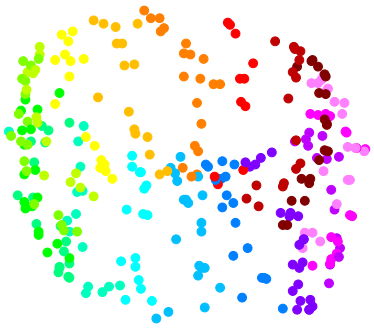
(a) Top view of embeddings produced by DrLIM



(b) Side view of embeddings produced by DrLIM

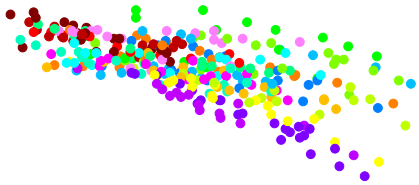


(c) Top view of embeddings produced by the proposed algorithm

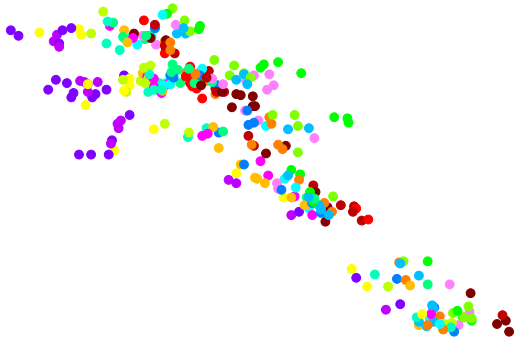


(d) Side view of embeddings produced by the proposed algorithm

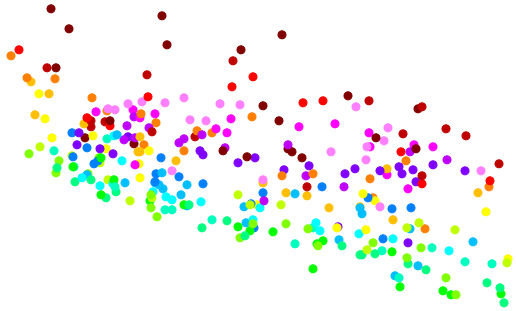
Figure 4.11: Embeddings produced by DrLIM and proposed model with full similarity (colours indicate varying azimuth).



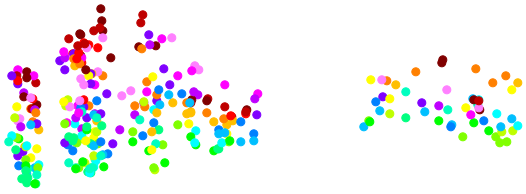
(a) First view of embeddings produced by DrLIM



(b) Second view of embeddings produced by DrLIM

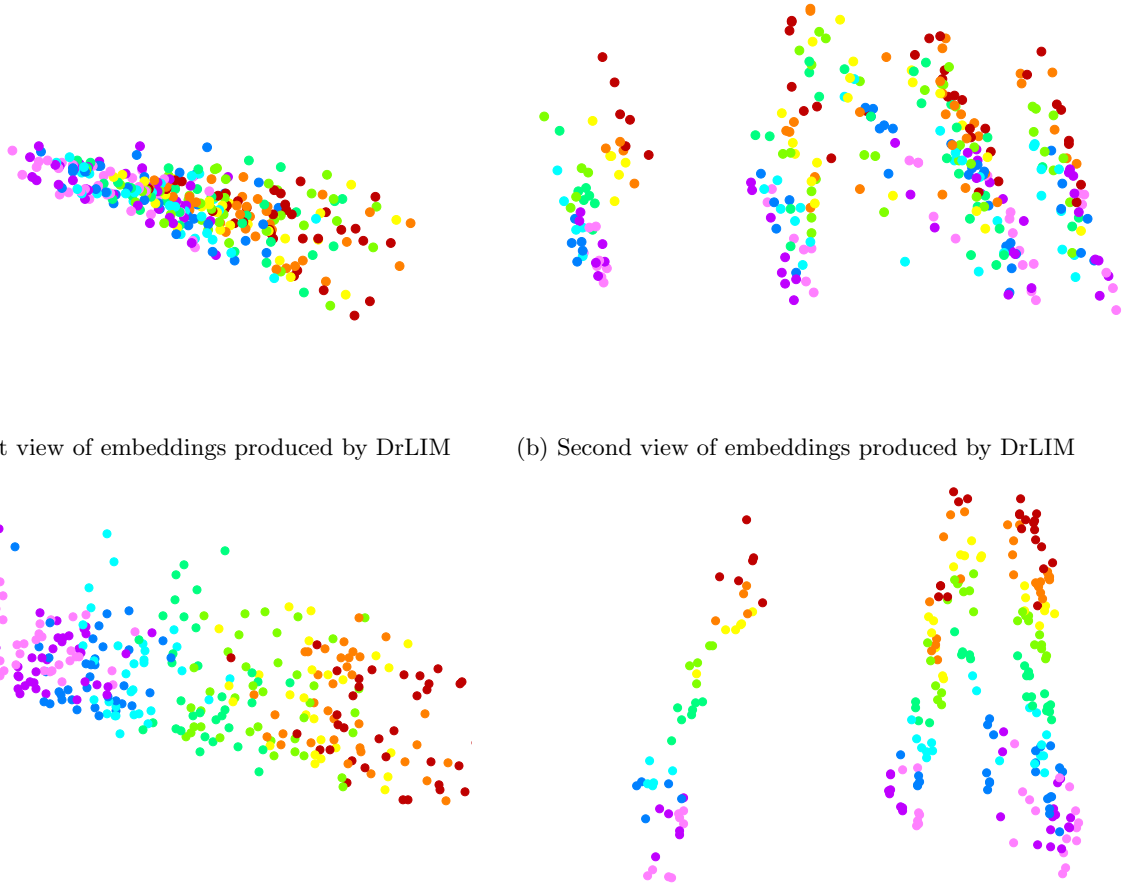


(c) First view of embeddings produced by the proposed algorithm



(d) Second view of embeddings produced by the proposed algorithm

Figure 4.12: Embeddings produced by DrLIM and the proposed model with chain-like similarity. Colours indicate varying azimuth.



(a) First view of embeddings produced by DrLIM

(b) Second view of embeddings produced by DrLIM

(c) First view of embeddings produced by the proposed algorithm

(d) Second view of embeddings produced by the proposed algorithm

Figure 4.13: Embeddings produced by DrLIM and proposed model with chain-like similarity. Colours indicate varying elevation.

Chapter 5

Conclusions and Future Work

Existing work on siamese architectures made use of unnormalized, hand-crafted cost functions. In this thesis, we formulated a probabilistic framework that maximized the likelihood of a binary similarity label and described the performance gains provided by such a similarity model. Considering the theoretical advantages of probabilistic over non-probabilistic cost functions, training siamese networks using parameterized, normalized (i.e., probabilistic) models of similarity can yield representations that are more useful for downstream classification tasks. We proposed several similarity models, and evaluated them based on theoretical analysis and experiments on real datasets. By comparing the proposed framework with state-of-the-art methods, we showed that the representations learned using probabilistic similarity models yielded competitive results on datasets including MNIST, AT&T ORL and COIL-100, with a simpler architecture. We also studied the proposed algorithm as a dimensionality reduction algorithm using the MNIST and NORB datasets. We found that the normalized probability model without hand-crafted costs was better for dimensionality reduction, and potentially more suitable for video-like data.

There are several issues that were not addressed in this thesis, including: What if the similarity labels are not binary? What would happen if the data contained noise? Under what conditions does the algorithm fail?

There are multiple approaches to address the first problem. If we are given similarity labels that are continuous, we can use threshold functions to convert the continuous labels to binary labels or use the softmax function to convert real value to a discrete set of levels. Then, the algorithm presented in this thesis can be directly applied. If we do not want to use threshold functions, then the cost function can be modified to include continuous similarity labels, for example by using Gaussian distributions.

For the second issue, in this work we have selected a heavy-tailed distribution as the conditional probability model. The heavy-tailed distribution is good at handling outliers, and a small amount of noise within images essentially creates more outliers. Therefore, we expect our proposed method to be able to handle a small amount of noise. Performance will likely degrade as the amount of noise increases.

Regarding to the third issue, as discussed previously, the proposed algorithm does not perform as well in terms of classification compared with dedicated algorithms when the amount of available data is small. Different datasets have different amounts of variations within classes. It is difficult to quantify the minimum amount of data required in order for the proposed algorithm to work well. The current implementation of the proposed algorithm does not perform as well as convolutional neural networks on datasets like Caltech101, CIFAR10, etc. However, this can be improved by changing the current

architecture to a convolutional network architecture, as well as to use a hierarchical model that can decomposes image features into parts, together with the proposed framework for applications related to datasets similar to the Caltech101 or CIFAR10 dataset.

The most impressive property of the proposed model is that it only requires weakly labelled data to achieved competitive results. In addition the model tends to work better with large number of training data. However, on the other hand, it is difficult to scale up the current model to work with data that consist of a large numbers of classes, since the proposed model solely relies on pairwise differences between images within/without classes. Thus, another possible direction is to incorporate unsupervised methods (such as an auto-encoder) as pre-trainning methods or as a part of an unsupervised objective.

Lastly, we did not investigate the possibility of using the learned representations for different down-stream tasks. For example, we have explored the capability of a siamese architecture if consecutive frames from videos of natural scenes were treated as similar in the context of dimensionality reduction, but we would still need to prove the point by conduct real experiments on video datasets.One possible approach is to use video data for object identification and incorporate the time-dependent similarity/dissimilarity for tracking and temporal coherence.

Appendix A

Effect of momentum on different datasets

There has been recent focus in the research community on developing momentum methods for accelerating gradient-descent convergence. In this appendix, we show several practical effects of regular momentum and nesterov momentum on different hidden units and datasets.

Figure A.1 shows a momentum study based on the MNIST dataset. The experiments were constructed using 20000 images from the training images of MNIST, a single hidden layer neural network with 250 sigmoid hidden units, and a discriminative cost function. The correct classification accuracies for 10000 validation images are shown. We use the 90% accuracy level as a reference to compare convergence speeds. Observe that nesterov momentum produced significant gains in the speed of convergence for sigmoid hidden units on the MNIST.

Although Figure A.1 show that nesterov momentum has significant gains on the MNIST dataset, Figure A.2 shows a momentum study on the AT&T face data. The experiments were constructed using 200 images from the AT&T faces dataset, and the same neural network architecture and cost function as the previous experiment. The accuracy for 200 validation images are shown. Clearly, nesterov momentum did not significantly improve the speed of convergence for sigmoid hidden units on this dataset. We further verified the data dependency of nesterov momentum on the COIL-100 dataset. The results are shown in Figure A.3 and are consistent with AT&T face data.

Finally, we studied the effect of momentum on different activation units. We performed the experiment on the MNIST dataset using the same set up as in Figure A.1, but with Relu hidden units. The results are shown in Figure A.4. There was a small but likely insignificant gain in convergence speed.

From the above experiments, we conclude that performance gains from the use of nesterov momentum are data dependent. However, we also observed that using momentum in gradient updates always resulted in a faster convergence speed than without momentum.

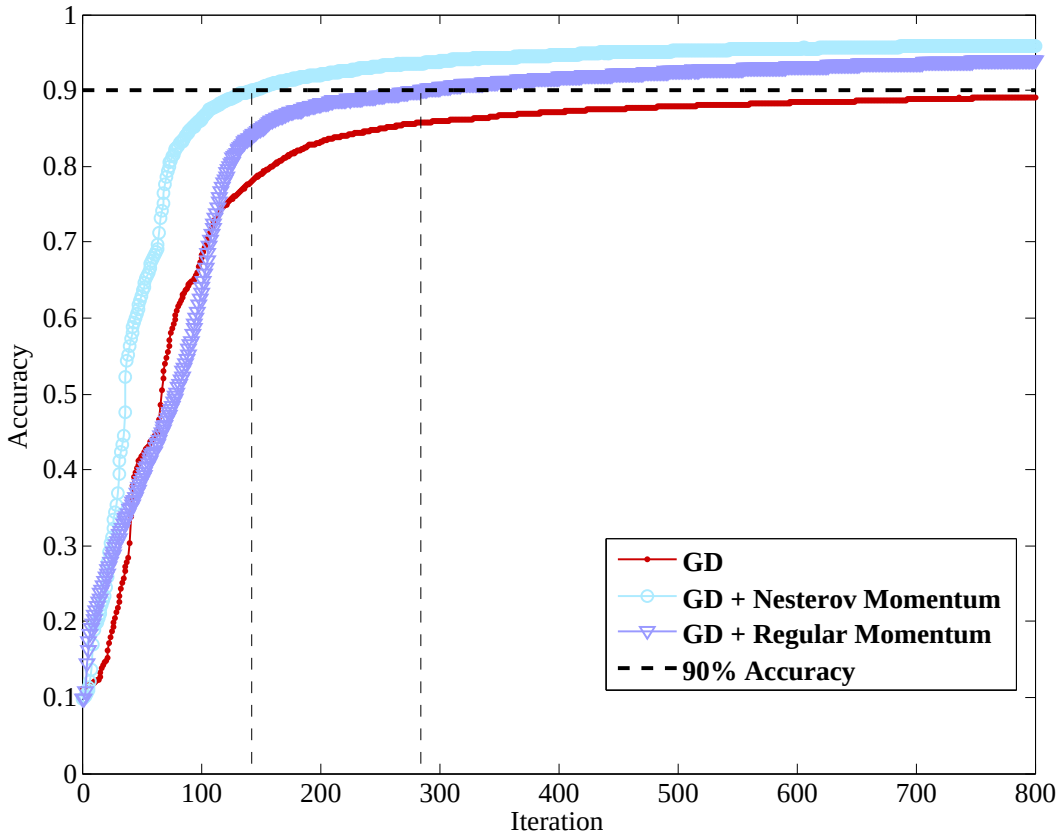


Figure A.1: Gradient Descent Update with and without momentum on MNIST

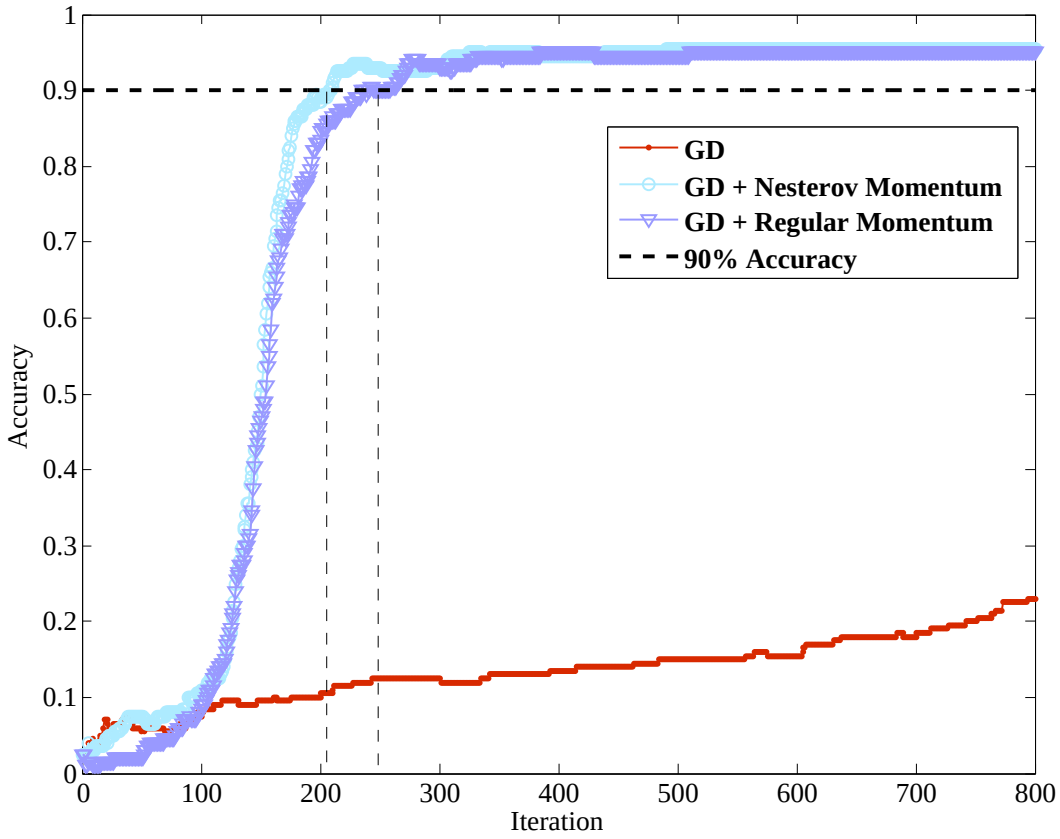


Figure A.2: Gradient Descent Update with and without momentum on AT&T faces

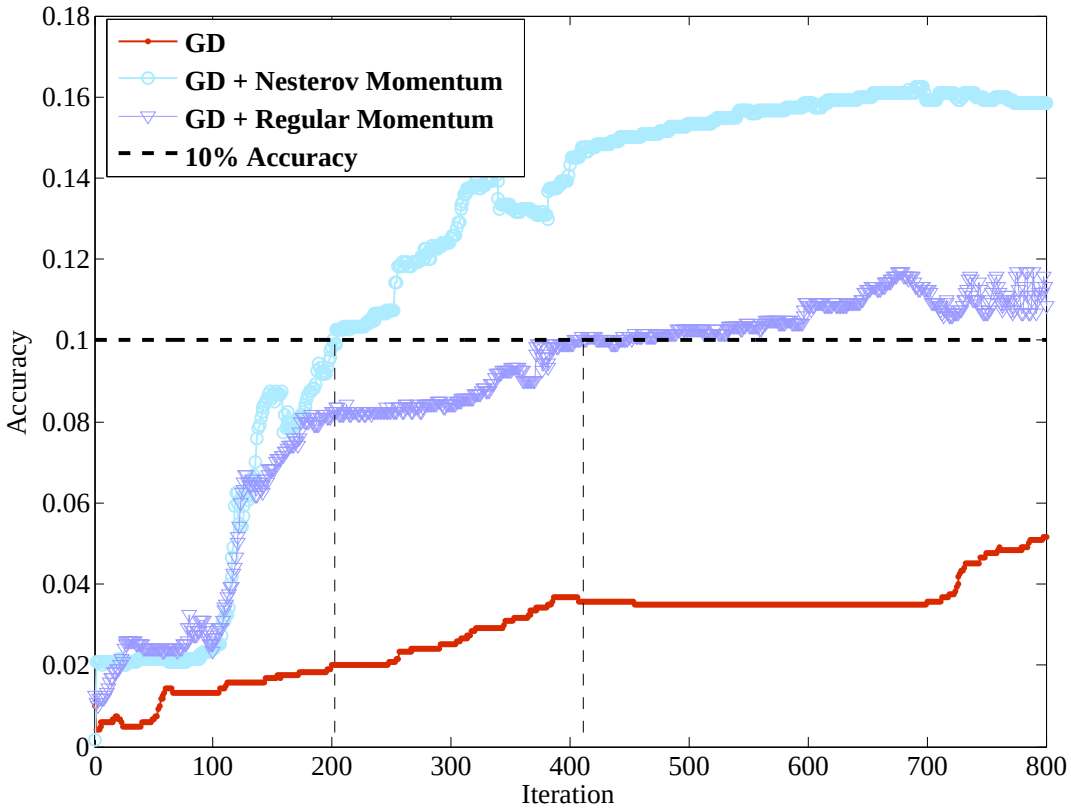


Figure A.3: Gradient Descent Update with and without momentum on COIL-100

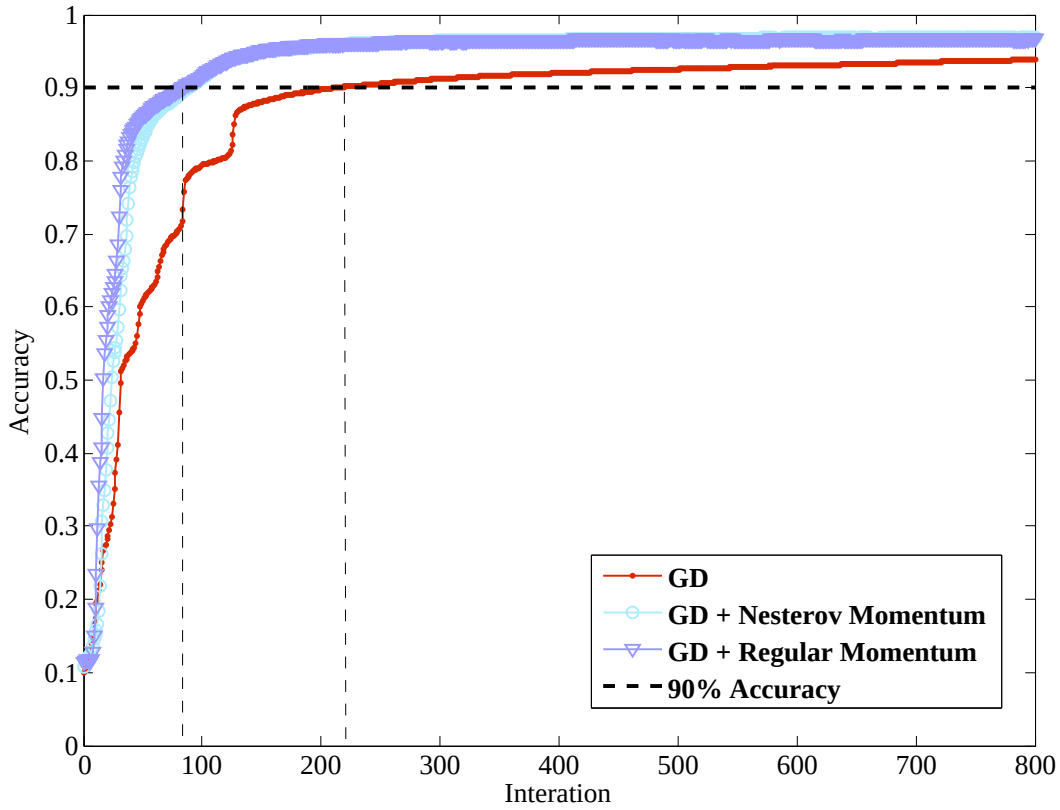


Figure A.4: Gradient Descent Update with and without momentum on MNIST. Relu hidden unit activation function.

Bibliography

[1] H. Larochelle, Y. Bengio, J. Louradour, and P. Lamblin, "Exploring strategies for training deep neural networks," *Journal of Machine Learning Research*, vol. 10, pp. 1– 40, June 2009.

[2] S. Becker and G. Hinton, "Self-organizing neural network that discovers surfaces in random-dot stereograms," *Nature*, vol. 355(6356), pp. 161–163, January 1992.

[3] J. Bromley, I. Guyon, Y. Lecun, E. Säckinger, and R. Shah, "Signature verification using a "siamese" time delay neural network," in *Proceedings of the Advances in Neural Information Processing Systems (NIPS)*, 1994.

[4] W.-T. Yih, K. Toutanova, J. Platt, and C. Meek, "Learning discriminative projections for text similarity measures," in *Proceedings of the Fifteenth Conference on Computational Natural Language Learning*, pp. 247–256, 2011.

[5] S. Chopra, R. Hadsell, and Y. LeCun, "Learning a similarity metric discriminatively, with application to face verification," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, vol. 1, pp. 539–546, June 2005.

[6] H. Mobahi, R. Collobert, and J. Weston, "Deep learning from temporal coherence in video," in *The International Conference on Machine Learning (ICML)*, 2009.

[7] K. Chen and A. Salman, "Extracting speaker-specific information with a regularized siamese deep network," in *Advances in Neural Information Processing Systems (NIPS)*, pp. 298–306, 2011.

[8] S. Lazebnik, C. Schmid, and J. Ponce, "Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, vol. 2, pp. 2169–2178, 2006.

[9] F.-F. Li, R. Fergus, and P. Perona, "Learning generative visual models from few training examples: An incremental bayesian approach tested on 101 object categories," 2004.

[10] S. Hoi, W. Liu, and S.-F. Chang, "Semi-supervised distance metric learning for collaborative image retrieval," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1–7, June 2008.

[11] K. Weinberger and L. Saul, "Distance metric learning for large margin nearest neighbour classification," *J. Mach. Learn. Res.*, vol. 10, pp. 207–244, February 2009.

[12] S. Xiang, F. Nie, and C. Zhang, "Learning a mahalanobis distance metric for data clustering and classification," *Pattern Recognition*, vol. 41, no. 12, pp. 3600–3612, 2008.

[13] R. Hadsell, S. Chopra, and Y. LeCun, "Dimensionality reduction by learning invariant mapping," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2006.

[14] O. Delalleau and Y. Bengio, "Parallel stochastic gradient descent," tech. rep., Talk presented at CIFAR NCAP Summer School, Toronto, Canada, 2007.

[15] Y. Nesterov, "Gradient methods for minimizing composite objective function, core discussion paper," Tech. Rep. 76, Center for Operations Research and Econometrics (CORE), Catholic Univ. Louvain, Louvain-la-Neuve, Belgium, 2007.

[16] G. Hinton, *Relaxation and its role in vision.* PhD thesis, 1978.

[17] C. Hu, J. Kwok, and W. Pan, "Accelerated gradient methods for stochastic optimization and online learning," in *Advances in Neural Information Processing Systems (NIPS)*, 2009.

[18] L. Xiao, "Dual averaging method for regularized stochastic learning and online optimization," in *Advances in Neural Information Processing Systems (NIPS)*, 2009.

[19] Y. Nesterov, "A method of solving a convex programming problem with convergence rate o(1/sqr(k))," 1983.

[20] I. Sutskever, *Training Recurrent Neural Networks.* Ph.d. thesis, University of Toronto, Toronto, Ontario, Canada, 2013.

[21] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, vol. 1, pp. 886–893, June 2005.

[22] G. David, "Distinctive image features from scale-invariant keypoints," in *International Journal of Computer Vision*, pp. 91–110, November 2004.

[23] B. Boser, I. Guyon, and V. Vapnik, "A training algorithm for optimal margin classifiers," in *the Fifth Annual ACM Workshop on Computational Learning Theory*, pp. 144–152, 1992.

[24] K.-W. Chang, C.-J. Hsieh, and C.-J. Lin, "Coordinate descent method for large-scale l2-loss linear support vector machines," in *Journal of Machine Learning Research*, vol. 9, pp. 1369–1398, 2008.

[25] F. Samaria and A. Harter, "Parameterisation of a stochastic model for human face identification," in *Proceedings of the Second IEEE Workshop on Applications of Computer Vision*, 1994.

[26] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," in *Proceedings of the IEEE*, vol. 86, pp. 2278–2324, November 1998.

[27] R. Salakhutdinov and G. Hinton, "Learning a nonlinear embedding by preserving class neighbourhood structure," in *Proceedings AI and Statistics*, 2007.

[28] S. Nene, S. Nayar, and H. Murase, "Columbia object image library (coil-100)," Tech. Rep. CUCS-006-96, Columbia University, February 1996.

[29] M. Yang, D. Roth, and N. Ahuja, "Learning to recognize 3d objects with snow," in *Proceedings of the Sixth European Conference on Computer Vision (ECCV)*, vol. 1842, pp. 439–454, 2000.

[30] Y. LeCun, F. Huang, and L. Bottou, "Learning methods for generic object recognition with invariance to pose and lighting," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, vol. 2, pp. 97–104, 2004.