Nonlinear Dimensionality Reduction Techniques and Their Application in Neural Networks

Student: Michael Hobbs
Supervisor: Amir Hesam Salavati
Professor: Amin Shokrollahi

7, June 2013

Overview

This project aims to study nonlinear dimensionality reduction algorithms and the extent to which they can improve classification. In other words, will we be able to classify input data reliably after we have reduced its dimensionality? Will the rate of successful classifications drop? or it is possible to have the best of both worlds and even improve the classification rate? In this project, we consider different feature extraction algorithms and investigate their performance in classification tasks using Support Vector Machines (SVMs) as the classifier. We consider supervised and unsupervised feature extraction methods. In the unsupervised approach studied in this work, the goal is to extract a few features while maintaining the ability to reconstruct the original data with high enough quality. In the supervised approach, we extract those features that enable us to achieve better performance in the classification task. Thus, although we might lose the ability to reconstruct the data from the features, we potentially achieve better classification rates.

1 Introduction

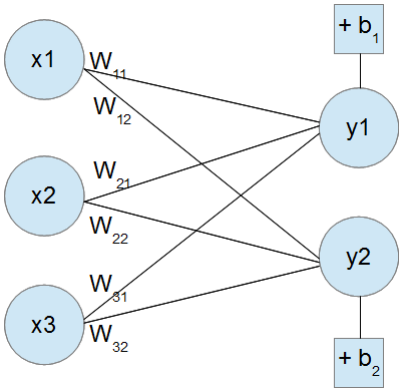
Pattern classification algorithms usually have to deal with high-dimensional data. As in other domains, the "curse of dimensionality" causes the algorithms to be slower and sometimes even not being able to achieve their objectives. For this reason, a "feature extraction" stage is often employed to reduce the dimensionality of the input data before applying the classification task. In this work we will focus on the case of images but everything we will discuss is applicable to other types of signals, unless explicitly described as due to the nature of images. Dimensionality reduction encompasses all the methods which provide a way to transform data from its original dimensionality D to a smaller dimensionality d.

In its simplest form feature selection consists of sub-sampling the original data, selecting the dimensions, or features, which are the most relevant or least redundant. However, more elaborate approaches also exist. Some such approaches are based on applying pre-defined filters to the data. The others "learn" these filters as well, ensuring that the features are selected in such a way that the original data can be reconstructed from the features reliably or the success rate of the classifier is increased.

In this work, we are concerned with the algorithms of the second category, i.e. those that learn which features to extract. This is achieved by learning a mapping between the original and the dimensionality-reduced data, which we refer to as the feature-set. This mapping can be linear or non-linear.

The linear mapping is captured in a matrix W that maps the D-dimensional input data "x" to a d-dimensional feature vector y, with $d << D$. More

specifically, $y = Wx + b$, where y is the $1 \times d$ feature vector, x is the $1 \times D$ data vector, b is the bias and W is the $D \times d$ mapping. Various algorithms exist, each with its advantages and pitfalls, PCA (principal component analysis)[19] is perhaps the most well-known linear feature extraction method.



Linear mappings can be easily extended to non-linear mappings by sending the feature vector y through a nonlinear function (such as tanh(.)). For instance, a linear mapping such as PCA can be transformed to a non-linear mapping by what is called the "kernel trick"[10] or simply applying a non-linear function to the feature vectors[5][11].

These mappings, whether linear or nonlinear, can be performed using a neural network. Well known machine learning algorithms that use neural networks include MLP (multi-layer perceptron)[9] and RBM (restricted boltzmann machines)[15]. Each has its strengths and weaknesses and is better suited to certain tasks than others. They also differ in the architecture of the neural network they use, e.g. single/multi layer, which affect the performance of the algorithm as well. Algorithms that use several layers are also known as "deep networks" and fall in the field of deep learning[1]. The optimal depth and architecture of a neural network is outside the scope of this project and is an aspect which could be worked on in order to further optimize the classifier.

In spite of their differences, all these approaches have a common part in that they learn the mapping matrix W by minimizing (maximizing) an objective function. There are different approaches for implementing this minimization (maximization) but most of them rely on adjusting the weights towards the desired direction. the most well-known of these approaches is the gradient descent (ascent) where the weights are updated based on the direction of the gradient. One can either calculate the gradient over the whole dataset first and then update the weights or calculate the gradient sample-by-sample and update the weights gradually. This later approach is known as stochastic gradient descent and is the focus of this work.

When an architecture has more than one layer, the gradient has to be sent back through the network starting at the final layer and working down to the first layer. This is called backpropagation[4].

Any algorithm using backpropagation can put aside a portion of the training dataset in order to perform early stopping[8]. This subset of the initial training set is called a validation set and its purpose is to avoid overfitting the network to the training data. Overfitting is when the network has adapted too thoroughly to the training set, thus representing the characteristics of the training set rather than the actual characteristics of the whole set of data which the network will have to classify. Early stopping consists of following the gradient of the training data set which attempts to minimize the error on the training set but setting an end condition of when the error computed on the validation set has started to increase, stopping thus as soon as the learning has started capturing training-set-specific features. For instance: if the algorithm is trained to identify "trees" in an image, overfitting corresponds to only distinguishing those trees that have already been seen as a "tree", preventing the algorithm to "generalize" what it has learned. On the other hand, we have underfitting, which corresponds to, for instance, assigning the label "tree" to any thing that is green.

The rest of this report is organized as follows. In Section 2 we will go over some related work. Section

3 is dedicated to explaining the methods we are interested in. Experimental results are discussed in Section 4. Finally, Section 5 concludes the report and sheds some light on some on-going and future work.

2 Related Work

2.1 Sparse Coding

Previous research on image recognition has involved learning a dictionary of features, each trained on small patches of the data (for example: a window of 6 pixels of an image), by an unsupervised algorithm (i.e. without information about the classes of the data), and then learning an encoding which passes all the small patches of an image through the dictionary resulting in a list of features, before finally learning a classifier on the features using class information (supervised learning). Coates and Ng[3] studied the performance of such a system using various algorithms for training the dictionary and training the encoder, including Sparse coding and auto-encoders with a high emphasis on the former.

To learn the dictionary, the Sparse coding algorithm uses coordinate descent to minimize, alternatively over the sparse codes $s^{(i)}$ and the dictionary D, the L1-penalized sparse coding formulation:

$$\min_{D,s^{(i)}} \sum_i \|Ds^{(i)} - x^{(i)}\|_2^2 + \lambda \|s^{(i)}\|_1$$
$$\text{subject to } \|D^{(j)}\|_2^2 = 1, \forall j$$

where $D^{(j)}$ is the j-th column of D and λ is the sparsity penalty coefficient.

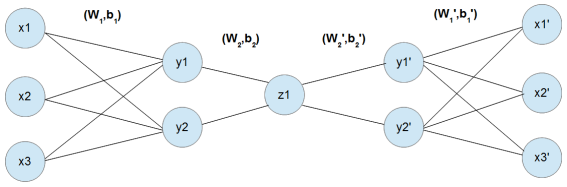
Learning the encoding is done by minimizing the same formulation while the dictionary is fixed. In terms of spare coding as encoder, Coates and Ng found that it excelled at preparing small datasets for the classifier, regardless of which method was used to make the dictionary, even on random dictionaries. However, in this work we are not interested in small datasets.

2.2 Auto-encoders

Auto-encoders aim to learn a manifold over a dataset in a coordinate system of a different dimensionality in order to capture the manifold on which lies the main variance of the data. The intermediate mapped vector y is the sigmoid of a linear mapping: $y = \sigma(Wx + 1)$ and y is then mapped back to the output vector $z = \sigma(W'y + b')$ in which the following constraint may be used: $W' = W^T$. This mapping is achieved by minimizing the average reconstruction error:

$$\{\theta^*\}, \{\theta'^*\} = \underset{\{\theta^*\},\{\theta'^*\}}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} L(x^{(i)}, z^{(i)})$$
$$\text{Where } \theta = \{W, b\}$$
$$\text{and } \sigma(x) = \frac{1}{1 + \exp^{-x}}$$

In the reconstruction error, L is a loss function many of which exist, such as squared error $L(x, z) = \|x - z\|^2$. It should also be noted that auto-encoders may be stacked to form a deep structure.



Auto-encoders can also be trained to learn the correct manifold, even if the data is noisy. This is achieved by the means of densoising auto-encoders [18]. In denoising auto-encoders, the input data is deliberately contaminated with noise to achieve robustness in feature extraction. It has been shown that denoising autoencoders can actually perform better than their standard counterparts[14] as they

avoid overfitting. Finally, one should also note that no class information is used during training and auto-encoders are unsupervised.

2.3 Mutual Information Maximization

The goal of MMI is to maximize the mutual information between the class labels and the feature vectors, i.e. try to extract those features that gives us more information about class labels. MMI[17] performs a linear transformation of the data but can easily be extended to become non-linear. This algorithm attempts to overcome some of the limitations of other algorithms (like PCA), namely by using higher-order statistics and not just second-order statistics. This is helped by the use of a more general criterion: the mutual information between class labels and the transformed data. Some issues MMI encounters lie in computational complexity, indeed, MMI requires all pair-wise interactions between training data samples as in its discrete form mutual information between transforms and classes is a function of the interactions between classes, the interaction between data of the same class and the interaction between all data.

The function which MMI maximizes is:

$$I(C, Y) = \sum_{c} \int_{y} p(c, y) \log \frac{p(c, y)}{P(c)p(y)} dy$$

$$\text{Where } P(c) = \int_{y} p(c, y) dy$$

Where C is the class label and Y is the feature vector.

2.4 Support Vector Machines

Whereas the previous algorithms have been concerned with encoding (i.e. finding transformations that extract features), SVM[13], in its original form, is designed to distinguish between two classes and finds the hyper-plane which best separates the training data of each of two classes.

SVM can be made non-linear through the "kernel trick" which consists of a non-linear mapping to a higher-dimensional space and performing the dot product over the data points.

3 Method

In this work, three approaches are implemented: auto-encoder and MMI for dimensionality reduction, and SVM for classification. The data we worked with was the CIFAR dataset, comprising ten classes, each with 5000 training images and 1000 testing images. Each image is a 32 pixel square, is in grayscale and is composed of 1024 pixels in total.

The idea behind this selection of algorithms was that auto-encoders would remove redundant information from the images and capture recurrent features in the classes. It has the advantage that a reverse mapping is also trained and that the transformation could be reversed, for example if one wanted to classify some images and store them using less data while still allowing the initial images to be reconstructed. The choice of MMI was to learn features in such a way as to increase the amount of information about the class the transformed data contains. It was expected that MMI would better prepare the data for classification.

For the classifier, we use SVM, extended to the multi-class scenario.The reasons behind this choice are, on the one hand, the ability to compare results with state of the art research and on the other hand to experiment with different implementations of the multi-class SVM.

3.1 Auto-encoder

The auto-encoder implementation used in this work is part of Laurens van der Maaten's (Delft University of Technology) *Dimensionality Reduction Toolbox*

[16] which is available online and is free to use, change, or redistribute for non-commercial purposes. In this implementation, four layers are used to create y, the feature vector, and four more layers are used to reconstruct x from y. We elected to reduce subsets of the CIFAR dataset to 21 and 900 dimensions. In the first layer the dimensionality is increased and in subsequent layers the dimensionality is reduced. This approach is flexible in allowing more features to become apparent in the hidden layers of the network and eventually make their way to the transformed version.

The network is initialized with random low weights of the order of 10^{-4}. Then, the network is pretrained using denoising auto-encoder (i.e. by adding noise to the data). To this effect, the data is sent through the network in small batches (more computationally efficient), the gradient to minimize the reconstruction error is calculated and the weights are updated and finally the MSE is computed between the original data and its reconstruction from the network, providing a check for convergence. After pretraining, the part of the network devoted to reconstructing x from dimensionality reduced y (i.e. the final four layers) are discarded. The first three of these are set to the transpose of their counterpart in the first half of the network, except the final layer which is set to zeros. In this way, the lower layers of the network, which are the furthest from the backpropagation of the gradient, are initialized in a favorable manner. Setting the final layer to zeros is essential, otherwise the network is a complete mirror function and further training will have no effect. The final step of this implementation is to perform backpropagation on the pretrained network, finetuning it.

3.2 Maximization of Mutual Information

The MMI is made in house, based on Torkkola [17] and was used to reduce CIFAR to 100 and 500 dimensions. It iterates over the training data, taking a configurable number of samples every epoch at random. This subset is then used to find the mutual information and its gradient, which in turn provides a basis for the derivation of the gradient of the weights. A subset can be taken because it will represent the whole dataset faithfully. It is even recommended to perform stochastic gradient ascent over mini-batches in order to reduce complexity. A rule of thumb is that 4000 pairs in the subset is a good compromise. This iteration continues until the mutual information starts to decrease.

The equations for the mutual information and its gradients from [17] are:

$$I_T = V_{IN} + V_{ALL} - 2V_{BTW}$$

Where:

$$\begin{aligned}
V_{IN} =& \sum_c \int_y p(c,y)^2 dy \\
=& \frac{1}{N^2} \sum_{p=1}^{N_c} \sum_{k=1}^{J_p} \sum_{l=1}^{J_p} G(y_{pk} - y_{pl}, 2\sigma^2 I) \\
V_{ALL} =& \sum_c \int_y p(c)^2 p(y)^2 dy \\
=& \frac{1}{N^2} \left(\sum_{p=1}^{N_c} \left(\frac{J_p}{N} \right)^2 \right) \sum_{k=1}^{N} \sum_{l=1}^{N} G(y_k - y_l, 2\sigma^2 I) \\
V_{BTW} =& \sum_c \int_y p(c,y)^2 P(c) p(y) dy \\
=& \frac{1}{N^2} \sum_{p=1}^{N_c} \frac{J_p}{N} \sum_{j=1}^{J_p} \sum_{k=1}^{N} G(y_{pj} - y_k, 2\sigma^2 I) \\
\frac{\delta I_T}{\delta y_i} =& \frac{\delta V_{IN}}{\delta y_i} + \frac{\delta V_{ALL}}{\delta y_i} - 2\frac{\delta V_{BTW}}{\delta y_i} \\
\frac{\delta}{\delta y_{ci}} V_{IN} =& \frac{1}{N^2 \sigma^2} \sum_{k=1}^{J_c} G(y_{ck} - y_{ci}, 2\sigma^2 I)(y_{ck} - y_{ci}) \\
\frac{\delta}{\delta y_{ci}} V_{ALL} =& \frac{1}{N^2 \sigma^2} \left(\sum_{p=1}^{N_c} \left(\frac{J_p}{N} \right)^2 \right) \\
& \cdot \sum_{k=1}^{N} G(y_k - y_i, 2\sigma^2 I)(y_k - y_i)
\end{aligned}$$

$$\frac{\delta}{\delta y_{ci}} V_{BTW} = \frac{1}{N^2\sigma^2} \sum_{p=1}^{N_c} \frac{J_p + J_c}{2N} \cdot \sum_{j=1}^{J_p} G(y_{pj} - y_{ci}, 2\sigma^2 I)(y_{pj} - y_{ci})$$

Parzen density estimation:

$$G(y, \Sigma) = \frac{1}{(2\pi)^{\frac{d}{2}} |\Sigma|^{\frac{1}{2}}} \exp\left(-\frac{1}{2} y^T \Sigma^{-1} y\right)$$

In order to update the weights:

$$\Delta W = \frac{\delta I}{\delta w} = \sum_{i=1}^{N} \frac{\delta I}{\delta y_i} \frac{\delta y_i}{\delta w}$$

Where: $\frac{\delta y}{\delta W} = x^T$

Under the constraint:

$$W^T W = I$$

The constraint on the weight is taken into account by updating it without constraint and then orthonormalizing it with Gram Schmidt orthonormalization. Each iteration involves creating the Parzen density estimation and the distance matrices for all sample pairs in the current batch. This is why complexity skyrockets when large training sets are used without this subsampling.

3.3 Support Vector Machines

Matlab has a predefined and highly parametrizable SVM function inherently. However, a major drawback of its implementation is that it does not support more than two classes of data (i.e. it will not create a classifier for more than two classes, it creates a yes/no separation of the space).

In order to support more than two classes, two SVM-training and two SVM-classifying implementations were devised. The first training method consists simply of creating a classifier for each class which learns to distinguish data of the class and data not from the class. The second training method consists of training a classifier "Class(1) vs. Not Class(1)" and then for every subsequent class training a basic SVM on that class against all the classes which do not yet have an SVM. The idea here is to not use the same information several times: when testing if one is classified as not belonging to the first class by the first SVM, then we no longer need to test belonging to the first class.

The first of the two testing functions for multi-SVM checks the data in all the trained SVMs one after another until it finds an SVM which successfully classifies it. The second checks all the trained SVMs exhaustively and applies the class whose SVM had the highest accuracy, thus giving the data the class which it most strongly fit. The second approach aims to avoid premature classification.

The underlying 2-way SVMs have the following model: linear kernel function, soft margin with adaptative box constraint, quadratic programming method and with an upper limit to the number of iterations.

3.4 Datasets and preprocessing

In the experiments the following datasets were used: CIFAR-10, Pima and Phoneme.

1. CIFAR-10 [2] was transformed to grayscale and vectorized and was subsampled in the experiments. Each image is comprised of 1024 pixels.

2. Phoneme [6]. It consists of 1962 train and 1962 test samples each of 20 dimensions and belonging to one of 20 classes. In this work, the classes were coded as follows: $A = 1$, $I = 2$, $\# = 3$, $N = 4$, $O = 5$, $M = 6$, $U = 7$, $S = 8$, $E = 9$, $D = 10$, $L = 11$, $[= 12$, $V = 13$, $J = 14$, $H = 15$, $\& = 16$, $Y = 17$, $R = 18$, $\backslash = 19$, and $F = 20$.

3. Pima [7] is a 2 class dataset and consists of 500 training samples and 200 testing samples. Each sample is 8-dimensional.

4 Results

Fig. 1 illustrates the misclassification rate on the Pima dataset when MMI encodes and our multi-SVM classifiers on it. MMI encoding involved using batches of 30 samples (around 1000 pair-wise interactions) and the number of iterations was bounded to 1000. In the case of a dataset consisting of only two classes,it appears that all the multi-SVM approaches were equivalent to the ordinary 2-way SVM. This is due to each multi-SVM method starting by training a 2-class SVM capable of distinguishing "Class 1" data from all the other data, i.e. both multi-SVM methods train the same classifier. $SVM^{(2)}_{test}$ gives the class whose SVM scores highest to the data, except for the last class, which is given when the final SVM scores negatively but higher in absolute value than all preceding SVMs. However, when there are only two classes it will simply attribute a class based on the sign of the data's score. $SVM^{(1)}_{test}$ will only check one 2-way SVM in the standard manner and this involves simply checking the sign of the SVM's transformation of the data. As such, all the multi-SVMs are indeed equivalent when the data comprises 2 classes.

Fig. 2 shows irregular behaviour for $SVM^{(2)}_{train}/SVM^{(2)}_{test}$ in the larger ds on Phenome data encoded by MMI (batches of 30, max epochs: 1000) and classified by our multi-SVM methods. In fact, training the SVM with each class against all subsequent classes and classifying with the method which attributes a class based on which classifier achieved the highest score encountered a severe problem when the last two classes were such that one was very far from all the other classes and the other was relatively close. In this case, either the last class or the penultimate class received a very high score and was attributed to the majority of the data. This can be seen in the

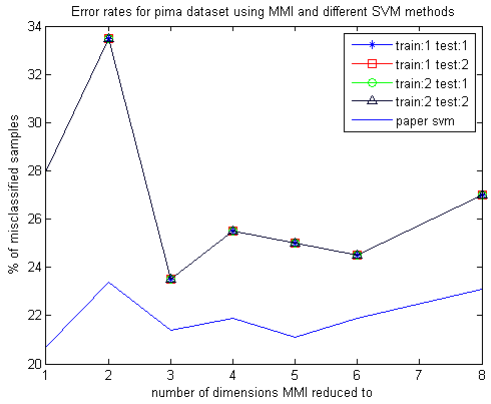


Figure 1: Pima MMI misclassification rates

"phoneme" results. A work-around would consist of reordering the classes but this severely reduces the robustness of this approach. The other test method for SVM was not affected since it attributed a class to the first SVM which "accepted" the data, this would occur before the final 2-way SVM was checked.

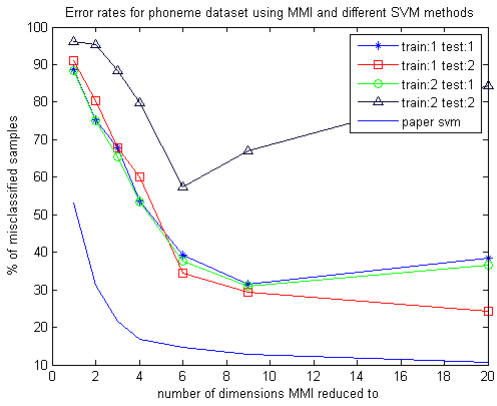


Figure 2: Phoneme MMI misclassification rates

It is also apparent in Fig. 2, that the multi-SVM train/classify pair which surpassed the others was training with $SVM^{(1)}_{train}$ and classifying with $SVM^{(2)}_{test}$. It coped better than the other methods especially when d, the dimensionality reduced to,

was close to D. Training with method 1 avoided the problems with one of the two last classes being far from the others by taking all the data into consideration when training the 2-way SVMs. Furthermore, the second test method outperformed the first test method because instead of labelling the data with the first class to "accept" it, it checked the extent to which the data was close to the classes and gave it the label of the closest class. It could be, for example, that the SVM corresponding to Class-3 vs. Other-Classes ended up with a hyper-plane with a slight overlap with a class whose SVM is yet to be trained. In this case, upon testing with method 1, some data in class, let's say 4, will be accepted by SVM-Class-3. Thus, this data will never get tested by SVM-Class-4 and will be misclassified.

Classification results from MMI encoding on the MMI confirmation sets, Pima and Phoneme, were slightly below those given in Torkkola [17]. This is due to an unoptimized selection of the σ Gaussian kernel variable. Indeed, no information can be found on how they chose it and so we chose to make it decay linearly from:

$$\frac{\max_{i,j} Dist(y_i, y_j)}{2}$$

to:

$$\frac{avg_{i,j} Dist(y_i, y_j)}{2}$$

All the same, our results follow the same trend as theirs confirming the discrepancy is simply due to a difference in choice of parameters, a sub-optimal SVM model and the fact we limit the number of epochs to 1000. It should be noted that the MMI is part of on-going work and that the results may be subject to improvement.

With respect to the CIFAR dataset, experiments performed on randomly generated subsets of 100 (Fig. 3) and 500 (Fig. 4) images per class, both for the train-set and for the test-set, resulted in very promising results. Size of the batches used in each iteration of the MMI had to be increased from that used for the "Pima" and "Phoneme" datasets, this is due to a vast increase in number of samples. Pima and Phoneme used batches of 30, giving around 1000 pair-wise calculations per batch, whereas CIFAR needed 60 samples per batch, around 4000 pair-wise calculations, in order to converge correctly. Like in the previous experiments, MMI was limited to 1000 epochs. It is interesting that the error rates achieved are competitive with state of the art results for classification of CIFAR-10 (which reach as much as 90.5% correct classification [12]) and suggests that the methods developed in this works may be worth pursuing. However, the high precision results may also be due to taking a random subset of the entire CIFAR dataset and most probably given we are working on grayscale images rather than color images (i.e. using lower dimensionality data with many fewer features).

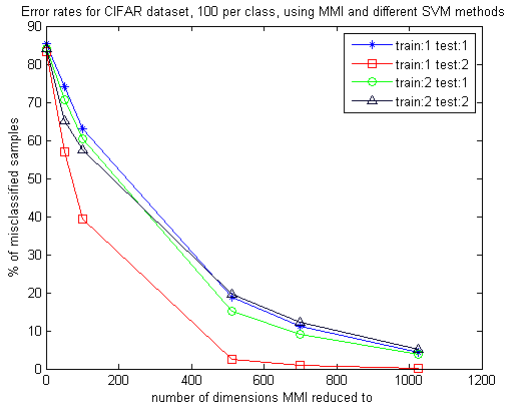


Figure 3: CIFAR(100) MMI misclassification rates

The auto-encoder, however, performed rather poorly in comparison. The lower classification rate is due to the algorithm's objective function: maximizing reconstructability. This objective function does not prepare data for classification tasks as well as the MMI, which aims to increase the mutual information between the transformation and class. Moreover, we didn't spend time on optimizing the auto-encoder and svm.

Again, the best SVM combination was

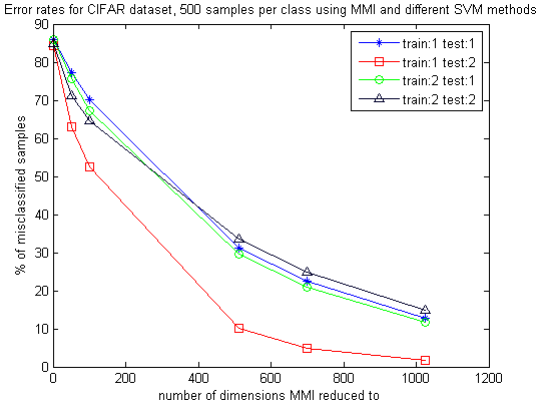


Figure 4: CIFAR(500) MMI misclassification rates

$SVM^{(1)}_{train}/SVM^{(2)}_{test}$, achieving up to 27.2% successful classification for d at 21 and 900. However, for the SVM other methods, success rates increases with d with rates ranging from 14.8% to 22.8%. Auto-encoder encoded a subset of the CIFAR set of 100 training images per class and 25 testing images per class, this may also have led to poorer performance.

In all cases, for all the CIFAR subsets, a successful classification rate of 100% was achieved when no dimensionality reduction was performed.

5 Conclusions and Future Work

We have seen that of all the methods studied in this work, the most promising is MMI with the multi-SVM: $SVM^{(1)}_{train}$-$SVM^{(2)}_{test}$. MMI could be improved by selecting pairs at random instead of batches from which pairs are made, the selected pairs would thus be able to span more of the initial data samples. The selection of σ could be studied and optimized. The size of the batches (or the number of pairs chosen per epoch) could be studied in function of the size of the dataset. A validation set could be used in order to perform early stopping and avoid any possible overfitting to the training-set (note that overfitting was not a problem in this work as the number of epochs was sufficiently bounded). Moreover, the MMI implementation could be made nonlinear which could potentially improve classification even further.

Furthermore, results suggest that dimensionality may be reduced by up to half while retaining an excellent classification rate and a perfect reconstructability (see: Fig.3).

We have also seen that the auto-encoder approach is outperformed and not suited to the goal of this work.

Finally, a very promising idea for future work is that of randomly subsampling each image by selecting small patches of a few pixels (let's say, to begin with, 5-10) and map each of these patches down to $d = 2$, 3, or 4. Then train a multi-SVM classifier for each patch and performing classification for each. Assign to the image the class to which the most of its patches were assigned. The patches could either be overlapping or spatially distinct. One could play around with the number of patches and the size of patches to optimize the performance of the classification method.

References

[1] Yoshua Bengio. Learning deep architectures for ai. *Foundations and Trends® in Machine Learning*, 2(1):1–127, 2009.

[2] CIFAR-10. `http://www.cs.toronto.edu/~kriz/cifar.html`.

[3] Adam Coates and Andrew Y Ng. The importance of encoding versus training with sparse coding and vector quantization. In *International conference on machine learning*, volume 8, page 10, 2011.

[4] Robert Hecht-Nielsen. Theory of the backpropagation neural network. In *Neural Networks, 1989. IJCNN., International Joint Conference on*, pages 593–605. IEEE, 1989.

[5] Erkki Oja. The nonlinear {PCA} learning rule in independent component analysis. *Neurocomputing*, 17(1):25 – 45, 1997.

[6] Phoneme. `http://www.cis.hut.fi/research/som_lvq_pak.shtml`.

[7] Pima. `http://archive.ics.uci.edu/ml/datasets`.

[8] Lutz Prechelt. Automatic early stopping using cross validation: quantifying the criteria. *Neural Networks*, 11(4):761–767, 1998.

[9] Dennis W Ruck, Steven K Rogers, and Matthew Kabrisky. Feature selection using a multilayer perceptron. *Journal of Neural Network Computing*, 2(2):40–48, 1990.

[10] Bernhard Scholkopf, Alexander Smola, and Klaus-Robert Mller. Kernel principal component analysis. In *ADVANCES IN KERNEL METHODS - SUPPORT VECTOR LEARNING*, pages 327–352. MIT Press, 1999.

[11] Matthias Scholz, Martin Fraunholz, and Joachim Selbig. Nonlinear principal component analysis: Neural network models and applications. In AlexanderN. Gorban, Balzs Kgl, DonaldC. Wunsch, and AndreiY. Zinovyev, editors, *Principal Manifolds for Data Visualization and Dimension Reduction*, volume 58 of *Lecture Notes in Computational Science and Enginee*, pages 44–67. Springer Berlin Heidelberg, 2008.

[12] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. *arXiv preprint arXiv:1206.2944*, 2012.

[13] Johan AK Suykens and Joos Vandewalle. Least squares support vector machine classifiers. *Neural processing letters*, 9(3):293–300, 1999.

[14] Chun Chet Tan and C Eswaran. Performance comparison of three types of autoencoder neural networks. In *Modeling & Simulation, 2008. AICMS 08. Second Asia International Conference on*, pages 213–218. IEEE, 2008.

[15] Tijmen Tieleman. Training restricted boltzmann machines using approximations to the likelihood gradient. In *Proceedings of the 25th international conference on Machine learning*, pages 1064–1071. ACM, 2008.

[16] Dimensionality Reduction Toolbox. `http://homepage.tudelft.nl/19j49/Matlab_Toolbox_for_Dimensionality_Reduction.html`.

[17] Kari Torkkola. Feature extraction by non parametric mutual information maximization. *The Journal of Machine Learning Research*, 3:1415–1438, 2003.

[18] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning*, pages 1096–1103. ACM, 2008.

[19] S. Wold, K. Esbensen, and P. Geladi. Principal component analysis. *Chemometrics and Intelligent Laboratory Systems*, 1987.