Methods and Applications for Distance Based ANN Training

Christoph Lassner, Rainer Lienhart
Multimedia Computing and Computer Vision Lab
Augsburg University, Universitätsstr. 6a, 86159 Augsburg, Germany
Email: {christoph.lassner, rainer.lienhart}@informatik.uni-augsburg.de

***Abstract*—Feature learning has the aim to take away the hassle of hand-designing features for machine learning tasks. Since the feature design process is tedious and requires a lot of experience, an automated solution is of great interest. However, an important problem in this field is that usually no objective values are available to fit a feature learning function to.**

Artificial Neural Networks are a sufficiently flexible tool for function approximation to be able to avoid this problem. We show how the error function of an ANN can be modified such that it works solely with objective distances instead of objective values. We derive the adjusted rules for backpropagation through networks with arbitrary depths and include practical considerations that must be taken into account to apply difference based learning successfully.

On all three benchmark datasets we use, linear SVMs trained on automatically learned ANN features outperform RBF kernel SVMs trained on the raw data. This can be achieved in a feature space with up to only a tenth of dimensions of the number of original data dimensions. We conclude our work with two experiments on distance based ANN training in two further fields: data visualization and outlier detection.

I. Introduction

Artificial Neural Networks are a flexible and powerful function approximation tool. Specifying the network topology allows to control the learning capacity, and specifying the error function and training method gives the possibility to control the learner's preference bias. While the usual choices are the mean squared error and stochastic gradient descent respectively, there are plenty of other options.

Replacing the mean squared error with a distance based error function, i.e. an error function that compares the distance of two mapped points in the codomain of the ANN to an objective value, opens up new application areas. As an immediate consequence, the absolute values of the learned mapping can vary, since the ANN only learns the shape of the objective function, but not its absolute position in the codomain. On the other hand, the different type of annotation used and this way of learning matches exactly the requirements of certain applications, such as feature learning, dimensionality reduction, outlier detection and visualization of high dimensional data.

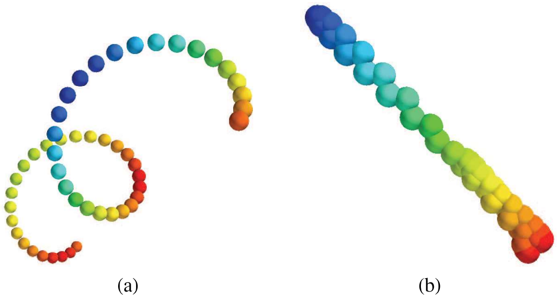

Fig. 1. A set of sample points and distance based learning results in 3D space. This simple but illustrative example demonstrates the idea of distance based ANN training. The points on the sine spiral in Figure 1a are located in 3D space. The colors are nonlinearly spread over the spiral. A neural network is trained to find a mapping to a 3D space where the euclidean distance between points corresponds to their color difference. Figure 1b shows the mapped points after one million steps of stochastic gradient descent.

The most important requirement in feature learning is to find a mapping from data space to a space where points that are considered to be similar for the target concept have a small distance, and points that are considered different have a large distance. Only these distance requirements are available, but no specific objective values for each input. However, an ANN trained on the objective distances can learn a feature space transformation. For an illustrative example, see Figure 1. Notice how easily a classifier can separate points with different colors in the learned feature space in Figure 1b with linear hyperplanes whereas this is not possible in the original space depicted in Figure 1a.

In this work, we briefly present methods and applications for distance based ANN training. After systematically deriving the rules for error backpropagation through networks of arbitrary depths, we generalize the method by introducing kernel functions. These naturally open up further application areas by making unsupervised training possible. In a series of experiments, we evaluate the performance of *DANN*s (ANNs trained on distances) for feature learning and dimensionality reduction, visualize the points of a 21-dimensional dataset and briefly introduce the use case of outlier detection.

II. Related Work

Constructing and training a neural network based on distances or similarities has, to the authors' knowledge, only been described in the paper "A similarity-based neural network for facial expression analysis" by Suzuki et al. [1]. In that work, the authors focus on facial expression analysis and only briefly introduce the necessary formulas for constructing their two layer network. They do not derive the general backpropagation formulas for networks of arbitrary depth and do not discuss general features and applications of distance based ANN training.

A DANN can be used, depending on the kernel function, in a supervised or unsupervised manner. In the case of an unsupervised learning scenario, it is similar to the autoencoder [2]. An autoencoder is an ANN with a specific structure (the amount of output nodes equals the amount of input nodes) that is trained to find a hidden representation that allows to reconstruct examples well. However, a DANN has a different inductive bias. For the standard kernel that is used in an unsupervised setting, the DANN is trained to find a hidden representation that preserves the original distances of the examples. This bias is particularly interesting in certain applications (especially in some dimensionality reduction scenarios and in data visualization), where it is a lot more appropriate than the one of the autoencoder. In all experiments where it is applicable, we provide the results of applying an autoencoder on the data to give a direct comparison of both methods.

Whereas an autoencoder can reconstruct data from the low-dimensional space using its decoding layers, the DANN can not. However, this comes with a theoretical speed gain during training by a factor of two (the DANN does not have a decoding layer, which corresponds to the application of a matrix of the same size of the encoding matrix). Since the reconstruction capability is not needed if just the transformed data representation is required, this advantage can be used in many cases.

The literature about feature learning focuses mostly on methods involving NNs or Convolutional Neural Networks with many layers, referred to as *deep learning*. These deep architectures have many hyperparameters and very long training times. Additionally, they have many parameters which leads to severe overfitting if no countermeasures are taken [3]. An analysis of ‘shallow’ architectures (few layers) in feature learning is given in [4] by Coates et al. Their results suggest that shallow architectures are competitive.

The de-facto standard for dimensionality reduction is the Principal Component Analysis (PCA). The comparison with this method is included in all our experiments where it is applicable. The DANN for dimensionality reduction is considered superior to Multidimensional Scaling (MDS), since MDS needs to use all points for scaling at training time, while a DANN can be applied to new data after training. This is an advantage for the proposed DANN method in the field of high-dimensional data visualization where MDS is often applied.

We only briefly introduce the possible application of outlier detection, but do not intend to compete with the state of the art in that field. We compare our method with a One-class SVM [5] and the fit of a thresholded elliptic envelope to give the reader a comparison with standard methods.

III. Distance based ANN training

The standard stochastic gradient descent algorithms require that the error function must be computable per example, and that it is differentiable. In the case of distance based training, the error function should capture the distance of

1) the results of forward propagation through the network (the distance in the codomain) and
2) a specified objective distance

for any pair of points (x_1, x_2).

To allow for the introduction of kernel functions, we denote the function for 1) as $\theta_f\left(x_1, x_2\right)$ and introduce a second function for 2) that calculates the objective distance $\theta_o\left(x_1, x_2\right)$. θ_o can, in the simplest case, evaluate to the user-annotated objective distance of x_1 and x_2.

In the following sections we will introduce a distance based error function that can be derived for all ANN connection weights. Hence, all stochastic gradient descent algorithms can be applied. The notation used is similar to [6]. The original notation is parameterless and assumes the propagation for only one example at a time (e.g. the output of node j in the network is denoted with o_j for *the current example*). Since we use pairs of examples, the values for the second example are marked with ’ (e.g. the output of node j for the second example is denoted with o'_j).

Let x_{ij} denote the ith input to network unit j, w_{ji} the weight associated with the edge from unit i to unit j and $net_j = \sum_i w_{ji} \cdot x_{ji}$ the weighted sum of inputs for unit j. Let the output computed by the jth unit of the ANN be denoted by o_j. With these preceding definitions, we define the error as

$$\begin{aligned} \mathrm{E}\left(x_1, x_2\right) &= \frac{1}{2}\left\|\theta_f\left(\boldsymbol{o}, \boldsymbol{o'}\right) - \theta_o\left(x_1, x_2\right)\right\|^2 \\ &= \frac{1}{2}\left(\theta_f\left(\boldsymbol{o}, \boldsymbol{o'}\right) - \theta_o\left(x_1, x_2\right)\right)^2 . \end{aligned} \tag{1}$$

The distance function for the network output is chosen as the squared Euclidean distance, i.e.

$$\theta_f\left(\boldsymbol{o}, \boldsymbol{o'}\right) = \frac{1}{2} \sum_{r \in outputs} \left(o_r - o'_r\right)^2 , \tag{2}$$

with *outputs* being the set of output units.

A. Derivation of the error function

To be able to apply gradient descent $\frac{\partial E(x_1, x_2)}{\partial w_{ji}}$ must be found for all i and j. If this term is known for all weights, e.g. standard stochastic gradient descent can be applied.

To specify these values for networks with arbitrary depths, it is necessary to make repeated use of the chain rule and build up an inductive definition. First, consider that any weight can only influence the error function through paths to output nodes of the network. Let the set of output nodes that can be reached from node j be denoted with $outlets(j)$. It is possible to split up

$$\frac{\partial E\left(x_1, x_2\right)}{\partial w_{ji}} = \sum_{r \in outlets(j)} \frac{\partial E\left(x_1, x_2\right)}{\partial\left(o_r - o'_r\right)} \cdot \frac{\partial\left(o_r - o'_r\right)}{\partial w_{ji}} \tag{3}$$

The first part of the equation, $\frac{\partial E(x_1, x_2)}{\partial(o_r - o'_r)}$, can be treated similarly for all edge weights, independent of their depth in the network. It evaluates to

$$\frac{\partial E\left(x_1, x_2\right)}{\partial\left(o_r - o'_r\right)} = \left(\theta_f\left(\boldsymbol{o}, \boldsymbol{o'}\right) - \theta_o\left(x_1, x_2\right)\right) \cdot \left(o_r - o'_r\right) \tag{4}$$

The second term in Equation 3 splits up as follows:

$$\frac{\partial\left(o_r - o'_r\right)}{\partial w_{ji}} = \frac{\partial o_r}{\partial net_j} \frac{\partial net_j}{w_{ji}} - \frac{\partial o'_r}{\partial net'_j} \cdot \frac{\partial net'_j}{w_{ji}} \tag{5}$$

This is the basis for the inductive definition. For a weight in the output layer of the network, the remaining terms become

$$\frac{\partial o_r}{\partial net_j} = h'\left(net_j\right), \; \frac{\partial o'_r}{\partial net'_j} = h'\left(net'_j\right), \tag{6}$$

where h' denotes the derivation of the applied transformation function and

$$\frac{\partial net_j}{w_{ji}} = x_{ji}, \; \frac{\partial net'_j}{w_{ji}} = x'_{ji}. \tag{7}$$

For a network unit j in a preceeding layer of the network, let $downstream(j)$ denote the set of units in the following layer unit j has a connection to. Then

$$\begin{aligned} \frac{\partial o_r}{\partial net_j} &= \sum_{k \in downstream(j)} \frac{\partial o_r}{\partial net_k} \cdot \frac{\partial net_k}{\partial net_j} \\ &= \sum_{k \in downstream(j)} \delta_k \cdot \frac{\partial net_k}{o_j} \cdot \frac{o_j}{\partial net_j} \\ &= \sum_{k \in downstream(j)} \delta_k \cdot w_{kj} \cdot h'\left(net_j\right) \end{aligned} \tag{8}$$

with corresponding results for $\frac{\partial o'_r}{\partial net'_j}$. The results from Equation 7 still apply.

Note that the choice of the symmetric distance function in Equation 2 is important for the symmetry property of the backpropagation algorithm (i.e. it does not matter if a pair of examples is used as (x_1, x_2) for training or as (x_2, x_1)). This is a usually wanted property. It also holds for the derivation (see Equations 4 and 5).

B. Example

Using distance based training, the network has the advantage that it can "place" the samples at well-fitting positions in its codomain as long as they have the appropriate pairwise distances. The network outputs can hence be located at very different positions in the codomain over several runs: they are equivalent for the error function under all distance-preserving transformations.

We implemented the distance based learning algorithm for a two layer network. A network of this depth can already approximate all continuous functions, which is the family of functions we intend to use for feature learning. To remove the step size parameter, we use an automatic step size estimation method described by Paweł Wawrzyński in "Fixed point method of step-size estimation for on-line neural network training" [7], leading to results superior to manual step size tuning.

Figure 1 shows the learning results on a simple example to illustrate the concept. The network learns a mapping from 3D to 3D space using the similarity annotations represented by the HSV color of the points. Similarly colored points should be mapped close together, whereas differently colored points should be mapped with greater distance. Note that it is not possible to use a linear hyperplane in the original space to classify differently colored points correctly, but it is possible in the learned "feature space" using the mapping of the DANN (depicted in Figure 1b).

C. Kernel functions

The distance function for the objective distance between two examples $\theta_o\left(x_1, x_2\right)$ occurs in the derivation function for all weights only in its original form (see Equation 4). This allows to extend its definition from objective distance annotations to arbitrary functions or a distance matrix. Theoretically, the distance matrix may even be non-symmetric.

To denote the general objective distance function, we introduce the notion of the kernel function similar to Support Vector Machines. Formally, the function $\theta_o\left(x_1, x_2\right)$ is replaced by the kernel function $\kappa\left(x_1, x_2\right)$ that returns a value representing the objective distance between the two input vectors. In contrast to SVMs, this function does not have to satisfy any axioms. However, it should be symmetric to facilitate learning. This is a property that holds for the most useful metrics. In the following list, we present the kernel functions that were implemented for our experiments. For these definitions, let $l(x)$ denote the class label of sample x.

$\kappa_{\text{Class}}\left(x_1, x_2\right) = \left\|l\left(x_1\right) - l\left(x_2\right)\right\|^2$

The objective distance is the squared distance in the annotation space. This function is a straightforward choice to make the feature mapping useful for a classification task. It is used to learn the mapping in Figure 1 with the HSV color as label.

$\kappa_{\text{Original}}\left(x_1, x_2\right) = \left\|x_1 - x_2\right\|^2$

The objective distance is equal to the original distance in the domain of the network. This function leads to a completely unsupervised learning method, similar to the autoencoder and MDS. In contrast to the autoencoder, however, it does not search for a latent representation of the data but tries to preserve the original distances. This is an appropriate inductive bias for many applications. Using this function, the DANN does not necessarily learn a representation of the data that is useful for a classification task.

$\kappa_{\text{Orig+Class}}\left(x_1, x_2\right) = \left\|x_1 - x_2\right\|^2 + \left\|l\left(x_1\right) - l\left(x_2\right)\right\|^2$

This kernel function combines the benefits of both of the aforementioned kernels. It makes the learner find a distance preserving transform including the class distance.

$\kappa_{\text{DPCA}}\left(x_1, x_2\right) = \left\|\text{DPCA}^{-1}\left(\text{DPCA}\left(x_1 - x_2\right)\right)\right\|^2$

The objective distance is equal to the distance reduced on the main differences between samples from the different classes (explanation follows).

The DPCA function denotes the projection of the difference vector into a PCA space (DPCA^{-1} denotes the reprojection). This space is learned before network training by sampling an arbitrary number of difference vectors between samples of different classes and applying PCA on them. The PCA hence learns the "principal differences" between the classes.

Projecting the vector into the PCA subspace and reprojecting it immediately yields the distance reduced to the main differences between the classes. Note that this kernel function has additional parameters: the amount of difference vectors for calculating the PCA and the amount of principal components to use.

TABLE I. Classification datasets and results

	thyroid	mushroom	gene
Dimensions	21	125	120
Classes	3	2	3
Samples (train/val/test)	3600/1800/1800	4062/2031/2031	1588/794/794
Balance	2.5%/95%/2.5%	52%/48%	25%/50%/25%
Lin. SVM	0.95	1.00	0.91
RBF SVM	0.97	1.00	0.91
Lin. SVM after DANN	**0.99**	**1.00**	**0.93**

IV. Applications and experiments

Due to their flexibility, DANNs can be used for a range of applications. In this section, we suggest certain setups and present an experiment for each suggested application area. The focus lies on feature learning and dimensionality reduction, where we give a comprehensive qualitative evaluation of results. Additionally, we address some practical pitfalls that must be taken into account to apply distance based learning successfully.

A. *Data preprocessing*

For all our experiments we apply the standard ANN preprocessing steps: the input data is normalized to have mean zero and standard deviation one. We then use an appropriately scaled $tanh$ transfer function in the first layer, as suggested by Le Cun et al. [8].

For distance based training it is even possible to apply a normalization on the annotation data as long as the relative distances are preserved. This facilitates learning, since the derivation function is more often evaluated at "steep" locations, and updates have more effect. We apply this method for preprocessing to make use of this advantage.

We noticed that it is critical to apply a feature selection process before training the ANN to remove noisy feature dimensions without or with few information about the target concept. To have a fair comparison, we apply this also as a preprocessing step for the autoencoder.

The feature preselection is automated by training a decision tree without any structural bias, i.e. no maximum depth and no minimum number of examples at leafs or nodes. This leads to a perfectly overfitted tree on the training data. From the tree's feature selections, we compute feature importances in accordance with the frequency of feature occurrence. For training the networks, we only use the most informative features that hold 90% of the total feature importance and discard the rest. As an alternative to using a decision tree as feature selector we experimented with an overfitted AdaBoost classifier that achieved comparable results.

B. *Feature learning and compression*

The first usage scenario we propose is automated feature learning and feature compression. A useful feature transformation for classification fulfills the following three requirements:

1) It maps input vectors with the same classification labels on close points in the codomain.
2) The distance between output values of input vectors with different labels is large.
3) Noise in the input vector is ignored.

We address point 3) with the automated preprocessing step. The points 1) and 2) are then tackled by using a distance trained ANN.

The number of output nodes (i.e. the dimension of the feature space) can be specified by the user and can be smaller than the dimension of the input space. If this is the case, the network also compresses the data. This can be very useful in a setting where large amounts of data are collected and should be saved for classification at a later point in time. Dependent on the used kernel function, the data can be compressed to be prepared for a classifier or to have as similar distances as in the original space as possible. This is not necessarily a choice that helps for a classification task, but can facilitate data analysis on the compressed data. The DANNs have in this case the advantage that

- they can project new data points quickly (compared to MDS) and
- they can use a nonlinear projection to the lower-dimensional space (compared to PCA).

1) Experimental setup: To give a detailed comparison of classification performance for various setups and give the reader the possibility to compare the DANN for different amounts of output nodes we designed the experiment for this use case as follows: for each dataset, we trained a DANN with ten different amounts of output nodes (feature space dimensions) with an equal step size in the range from one to one more than the number of original data dimensions.

To find the best amount of hidden nodes, we increase the amount of hidden nodes step-wise for each configuration until the error on the validation set does not decrease any more. Similarly, we use the validation set performance as early-stopping criterion to determine the amount of stochastic gradient descent steps.

The network weights are initialized by fan-in. To account for the slightly varying results of stochastic gradient descent and the random weight initialization, each experiment is conducted seven times and the best result (on the validation set) is used.

In the learned feature space, we apply a linear SVM classifier. The implementation used is LibLinear [9]. To have a fair comparison of SVM performance, the C parameter is evaluated for each SVM at each point in the range from 10^{-3} up to 10^{3}. The performance is evaluated after every 10 epochs (the "strip-length") of DANN stochastic gradient descent with a maximum amount of 120 strips. This upper limit is hardly ever reached.

We apply this setup on three datasets of the *Proben1* ANN benchmark set [10], in the version that is published together with the FANN library [11]. The training part of the data is used unchanged, the test data is split into validation and test parts with equal size (first and second half respectively). An overview over the datasets' characteristics can be found in Table I.

Since the *thyroid* dataset is very unbalanced, an evaluation metric must be used that truly reflects classifier performance and does not allow a classifier to 'ignore' a class. To be able to use the same metric for all experiments, including

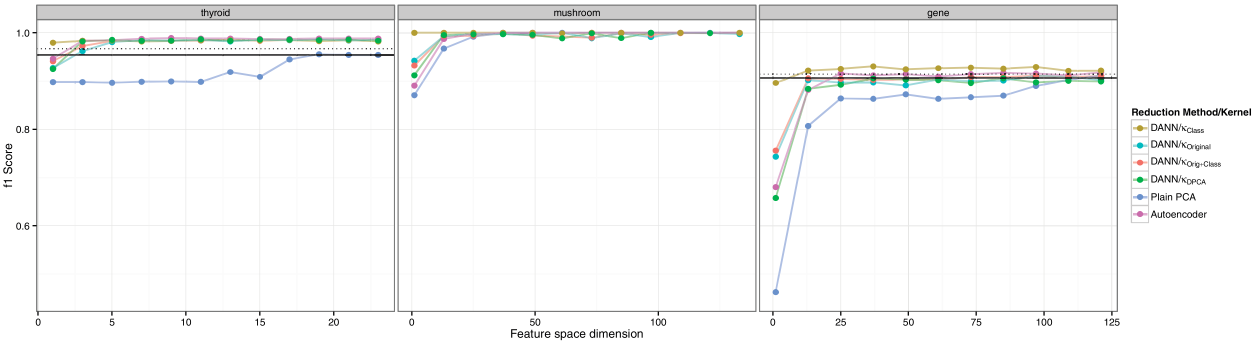

Fig. 2. Linear SVM classification results for automatically learned feature spaces. The black solid and dashed lines show the linear SVM and RBF kernel SVM on the original data with the full amount of dimensions.

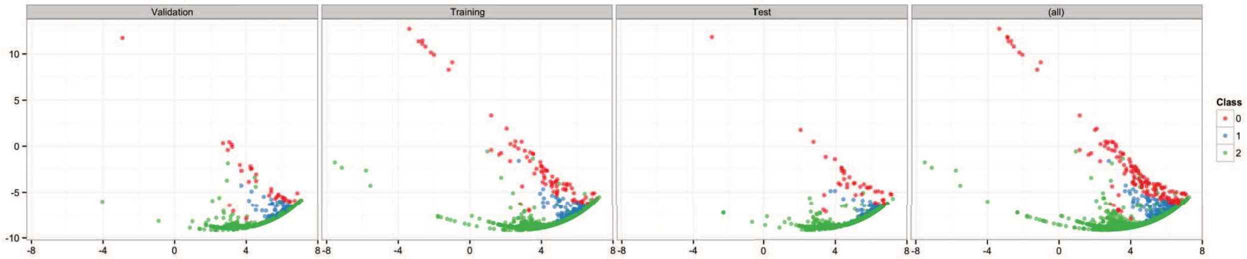

Fig. 3. Visualization of the *thyroid* dataset using a DANN with two output nodes and the κ_{Original} function. The network was trained only on the training set.

the ones with three classes, we use the f1 score. To have a fair performance comparison, the reference values of linear and RBF kernel SVMs trained on the original data (marked in Figure 2 by the black lines) have been determined with a parameter sweep for C and γ.

2) Results: The results of the experiment are summed up in Figure 2. The results on the *thyroid* dataset show the potential of the presented method: the classification performance can be increased with all kernels, even for a significantly lower amount of feature space dimensions. The performance of the RBF kernel SVM on the original 21 dimensions is still surpassed using the κ_{Class} kernel with one dimension, and for all other kernels except κ_{Original} with three dimensions.

The second part of the experiment is performed on the *mushroom* dataset to show that linear separability can be preserved with the feature space transformation. Hence we did not apply the feature selection preprocessing step to maintain it before performing the DANN learning. Both, the linear and RBF kernel SVM, reach 1.0 performance on the test set. The lines are omitted in the graphic to show the DANN results. For a dimension of 13, all kernels still achieve near 1.0 performance, the worst being κ_{Original} with an f1 score of 0.99.

We conclude the discussion of this experiment with an analysis of the results on the *gene* dataset. On this dataset, the difference between the different kernel functions becomes more apparent. The κ_{Class} kernel outperforms all others for the classification task, as expected. It can keep its performance as the only kernel up to a feature space dimension of one. The other kernels and the latent representation of the autoencoder stay en par with the SVM results up to a feature space dimension of 13. Only the κ_{DPCA} kernel performs worse than the others. The plain PCA fails in this dataset to capture the dimensions that are important for classification.

Summing up the results, we show that the performance of a linear and even an RBF kernel SVM can be outperformed consistently up to a very low amount of feature space dimensions on the analyzed datasets. The DANN method is en par with or outperforms PCA and the autoencoder in all experiments. In the following two sections, we demonstrate the performance of DANN learning in two further scenarios.

C. Data Visualization

The κ_{Original} kernel function opens up unique applications for data visualization. This kernel function allows to train a DANN in an unsupervised manner to find a mapping to a 2D (or 3D) space for visualization that preserves the distances from the higher dimensional space.

This is different from the PCA and the autoencoder, since these methods find a mapping to a lower-dimensional space that is decoupled from the original distances. It has advantages towards MDS since it allows to project points to the lower dimensional space at a later point in time. This makes visualization of data possible in settings where it was not possible before.

To give an illustrative example, we used a DANN with an intermediate layer size of eight nodes and trained it solely on the training data of the *thyroid* dataset. Figure 3 shows the

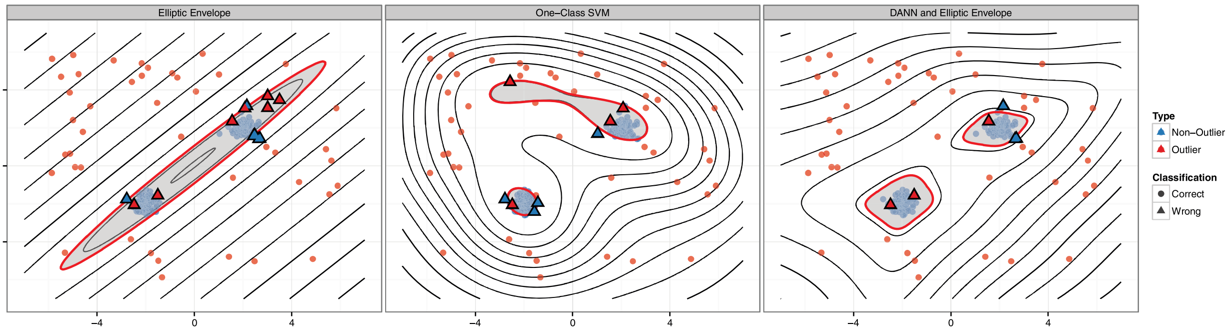

Fig. 4. Outlier detection results. With a DANN as preprocessing step, the elliptic envelope method can outperform an RBF kernel One-Class SVM.

projections of the different partitions of the data as well as the joined projections of all points. Even though the network was only trained on a part of the data it is used with, it projects all parts of the set to reasonable positions.

D. Outlier Detection

The last suggested use case is outlier detection. We exploit a natural bias of DANNs, namely the fact that during training the network will first learn to map clusters of points close together very accurately, whereas points that do not have many close neighbors are only learned later to be mapped to the right spot. This facilitates the identification of clusters of points and areas with many data points during early stages of DANN training.

We suggest to apply the DANN as a preprocessing step for an outlier detection method, such as a robust covariance estimator (e.g. an elliptic envelope) or a One-Class SVM. The DANN can be used to learn a mapping to a higher dimensional space in an unsupervised manner. This makes it easier for the succeedingly applied outlier detector to separate outliers from non-outliers, similar as an SVM projects the data to a higher dimensional space to separate it easier. The learning process must be stopped after the first epochs to use the aforementioned property.

It is straightforward to apply this approach in a partially supervised setting where there is no outlier annotation available but for a very limited set of data. We show an illustrative example in Figure 4, where we use a DANN as preprocessing step for an elliptic envelope method and demonstrate that this combination can outperform a One-Class SVM. For this experiment, we generated an additional set of annotated samples. All learners were trained in an unsupervised manner on the unlabeled data. Their parameters were estimated based on the classification results on the labeled data. The DANN training was stopped after only three epochs. With only five misclassifications, the combined classifier of DANN and elliptic envelope outperform the One-Class SVM.

V. Summary and Conclusion

Distance based ANN training is a versatile method that has so far hardly been described in literature. We derived an inductive definition of the distance error gradient for connection weights in layers of arbitrary depth and introduced the notion of kernel functions in this context.

The great flexibility that comes with the use of kernel functions opens up many application areas. We presented four, namely feature learning, compression, data visualization and outlier detection, and mentioned the specific advantages and possible pitfalls when using DANNs. The results in feature learning show that DANNs as feature learners coupled with fast, linear SVMs can outperform a finely tuned RBF kernel SVM.

Especially the combination of compressed feature extraction and classification at a later point in time makes DANNs attractive for applications with corresponding requirements. Similarly, DANNs have unique features for data visualization by learning a nonlinear distance preserving mapping to a low dimensional space with the possibility to map additional points after learning completed. Possible applications for outlier detection look promising, but a detailed evaluation remains to be done in future work.

References

[1] K. Suzuki, H. Yamada, and S. Hashimoto, “A similarity-based neural network for facial expression analysis,” *Pattern Recognition Letters*, vol. 28, no. 9, pp. 1104 – 1111, jul 2007.

[2] G. E. Hinton and R. R. Salakhutdinov, “Reducing the dimensionality of data with neural networks,” *Science*, vol. 313, no. 5786, pp. 504 – 507, July 2006.

[3] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” *NIPS*, 2012.

[4] A. Coates, H. Lee, and A. Y. Ng, “An analysis of single-layer networks in unsupervised feature learning,” in *AISTATS*, 2011.

[5] B. Schölkopf, J. C. Platt, J. C. Shawe-Taylor, A. J. Smola, and R. C. Williamson, “Estimating the support of a high-dimensional distribution,” *Neural Comput.*, vol. 13, no. 7, pp. 1443–1471, Jul. 2001.

[6] T. M. Mitchell, *Machine Learning*. McGraw-Hill, 1997.

[7] P. Wawrzyński, “Fixed point method of step-size estimation for on-line neural network training,” in *Proceedings of WCCI 2010 IEEE World Congress on Computational Intelligence*, July 2010, pp. 2012–2017.

[8] Y. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller, “Efficient BackProp,” *Lecture Notes in Computer Science*, vol. 1524, pp. 9 – 50, 1998.

[9] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin, “LIBLINEAR: A library for large linear classification,” *Journal of Machine Learning Research*, vol. 9, pp. 1871–1874, 2008.

[10] L. Prechelt, “PROBEN1 – A Set of Benchmarks and Benchmarking Rules for Neural Network Training Algorithms,” Fakultät für Informatik, Universität Karlsruhe, Tech. Rep., 1994.

[11] S. Nissen, “Implementation of a fast artificial neural network library (fann),” Department of Computer Science University of Copenhagen (DIKU), Tech. Rep., 2003, http://fann.sf.net.