Optimally Spaced Autoencoder

Daw-Chih Liou   Cheng-Yuan Liou

Department of Computer Science and Information Engineering
National Taiwan University
Taiwan, R.O.C.
cyliou@csie.ntu.edu.tw

***Abstract*— This paper presents a method for image restoration that uses a new object function for a multilayer perceptron (MLP) network. The training algorithm of the MLP aims to maximize the separation between patterns from different classes, while minimizing the distances between patterns from the same class. The trained MLP serves as a transformation encoder, mapping the pattern space into a new space where patterns from different classes are distinctly separated. This encoder accomplishes the 'optimally spaced coding' directly. It will enable effective resolution of challenging classification problems and restoration of severely corrupted images.**

Keywords*—*pattern recognition; classification; image restoration; vision; character recognition, optimally spaced codes

1. Introduction

The kernel function introduced in SVM [2] is a manually designed encoder. It transforms the space of a collection of patterns into a higher dimensional space. It is expected that a better discrimination, with certain degree of penalty, could be accomplished with such encoder by using a single perceptron. It is designed, originally, for two-class classification. The capsule network [7] could accommodate the abstract attributes of the whole image. The variations of a digit can be recognized, visually, by perturbing its trained attributes. The Bi-perceptron algorithm [3] accomplishes perfect classification, 100% correction, for any training dataset without any transformation encoder and gets high performance on testing dataset. The tiling algorithm [4] cannot tolerate any noisy pattern. This work further devises a new object function, or cost function, for the MLP that can facilitate and resolve the classification. The trained MLP is itself an ideal kernel function for a collection of images. With abstract attributes saved in neural weights from the whole image, it can tolerate noisy images.

2 Single-layer perceptron encoder

We formulate the training algorithm for a single layer perceptron. Assume that the values of input units can only be –1 or 1. In order to obtain distinct output codes for different class patterns, we redesign the object function for the MLP. This new object function is the overall inconsistence with the goal, named 'repellence energy'. Write,

$$E^{rep} = -\frac{1}{2}\sum_{p_1}^{P}\sum_{p_2}^{P}(d(y^{(p_1)}, y^{(p_2)}))^2 = \sum_{p_1}^{P}\sum_{p_2}^{P}E_{p_1p_2}.$$

Set $$E^{rep} = -\frac{1}{2}\sum_{p_1}^{P}\sum_{p_2}^{P}\sum_{m=1}^{M}(y_m^{(p_1)} - y_m^{(p_2)})^2, \quad (1)$$

where $y_m^{(p_1)}$ or $y_m^{(p_2)}$ is an M-dimensional output representation (code) corresponding to the p_1 or p_2 pattern. M is the number of neurons in the single layer. This repellence energy will force the representations to evolve in an M-dimensional hypercube space. Instead of the Hamming distance, the Euclidean distance is used as the distance function d in the derivation. Given P input patterns $\{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(P)}\}$, where the r^{th} pattern $\mathbf{x}^{(r)} = [x_1^{(r)} x_2^{(r)} \ldots x_N^{(r)}]$ is an N-tuple bipolar binary vector. In this case, each pattern belongs to a single class. There are total K classes. The vector $\mathbf{y}^{(r)}$ is the output vector of the layer neurons corresponding to the input pattern $\mathbf{x}^{(r)}$. The goal is to maximize the distance between every pair of output representations of different classes such that each class patterns is isolated from all other class patterns as far as possible. The overall distance is indicated by the total value of the energy. This energy is different from the covariance hypothesis introduced in [1] [8] and this encoder is different from those in [5].

To meet this goal, we reduce the energy $\boldsymbol{E}$ by means of the gradient descent rule [6] [9]. The algorithm for adjusting the neuron weights to decrease this energy is in below.

By differentiation, the gradient descent of the energy $E_{p_1p_2}$ is

$$\frac{\partial E_{p_1 p_2}}{\partial w_{ij}} = -\sum_{m=1}^{M}\left(y_m^{(p_1)} - y_m^{(p_2)}\right)\left(\frac{\partial y_m^{(p_1)}}{\partial w_{ij}} - \frac{\partial y_m^{(p_2)}}{\partial w_{ij}}\right)$$

$$= -\sum_{m=1}^{M}\left(y_m^{(p_1)} - y_m^{(p_2)}\right)\left(\frac{\partial f(net_m^{(p_1)})}{\partial (net_m^{(p_1)})}\frac{\partial (net_m^{(p_1)})}{\partial w_{ij}} - \frac{\partial f(net_m^{(p_2)})}{\partial (net_m^{(p_2)})}\frac{\partial (net_m^{(p_2)})}{\partial w_{ij}}\right)$$

$$= -\frac{1}{2}\left(y_i^{(p_1)} - y_i^{(p_2)}\right)\left\{\left(1-\left(y_i^{(p_1)}\right)^2\right)x_j^{(p_1)} - \left(1-\left(y_i^{(p_2)}\right)^2\right)x_j^{(p_2)}\right\} \quad ,$$

where

$$y_i = f(net_i), \quad net_i = \sum_{j=1}^{N} w_{ij} x_j, \text{ and}$$

$$f(net_i) = \tanh(0.5\, net_i) = \frac{1-\exp(-net_i)}{1+\exp(-net_i)}.$$

Note that

$$\frac{\partial (net_m^{(p_1)})}{\partial w_{ij}} = \frac{\partial (net_m^{(p_2)})}{\partial w_{ij}} = 0; \quad for \quad m \neq i.$$

The updating equations for the weights are

$$w_{ij} \leftarrow w_{ij} - \eta \frac{\partial E}{\partial w_{ij}}, \qquad (2)$$

where η is a positive learning constant. The threshold values $w_{i(N+1)}$ are updated in exactly the same way as that for the weights. Their updating equations are

$$w_{i(N+1)} \leftarrow w_{i(N+1)} + \frac{\eta}{2}(y_i^{(p_1)} - y_i^{(p_2)})((y_i^{(p_1)})^2 - (y_i^{(p_2)})^2),$$

and the fixed input is of value x_{N+1} = -1.

The initial weights may be set as $\boldsymbol{w}_{ij}$ = 0 for all i≠j and $\boldsymbol{w}_{ij}$ = 1 for i = j. These are orthogonal weights. Note that all the patterns will map (encode) to themselves using these weights. We then feed patterns one by one into the network and save their corresponding output vectors in an array. We calculate the Euclidean distance between every pair of output vectors of different classes. We use a square matrix ***D*** to store these distances. The value of its entry ***D****{i,j}* is the distance, $d(y^{(p_i)}, y^{(p_j)})$, between the output vector $y^{(p_i)}$ and the output vector $y^{(p_j)}$ (in response to the p_i pattern and the p_j pattern). Thus, the distance matrix ***D*** is symmetric and has zeros in all its diagonal entries. Among all the pairs of output vectors, we pick one pair that has the minimum distance. Then we use this pair of output vectors (indexed as $\boldsymbol{p}^{\boldsymbol{min}}{}_1$ and $\boldsymbol{p}^{\boldsymbol{min}}{}_2$) together with their corresponding patterns in the updating (2) to increase their distance.

For the next iteration, we feed all the patterns into the updated network again. We update the distance matrix ***D*** and increase the minimum distance. We repeat this batch procedure until the minimum distance cannot be increased or it is greater than a predetermined value.

Assume the patterns belong to K classes $\boldsymbol{X}_1$, $\boldsymbol{X}_2$, … $\boldsymbol{X}_K$, where class $\boldsymbol{X}_k$ contains P_k patterns { $\boldsymbol{x}^{(1)}$, $\boldsymbol{x}^{(2)}$, …, $\boldsymbol{x}^{(\boldsymbol{P}k)}$ }. The goal is to maximize the distance between every pair of output vectors that belong to different classes and minimize the distance between every pair of output vectors that belong to the same class. To meet this goal, we also need an object function that can provide the attraction energy for the same class patterns. This can be done by reversing the sign of the repellence energy, (1). The devised attraction object function for the same class patterns is in below.

The devised attraction energy is

$$E^{att} = \frac{1}{2}\sum_{p_k^1}^{P_k}\sum_{p_k^2}^{P_k}(d(y^{(p_k^1)}, y^{(p_k^2)}))^2 = \sum_{p_k^1}^{P_k}\sum_{p_k^2}^{P_k} E_{p_k^1 p_k^2}\,; \qquad (3)$$

and

$$\frac{\partial E_{p_k^1 p_k^2}}{\partial w_{ij}} = (y_i^{(p_k^1)} - y_i^{(p_k^2)})\{(1-(y_i^{(p_k^1)})^2)x_j^{(p_k^1)} - (1-(y_i^{(p_k^2)})^2)x_j^{(p_k^2)}\},$$

(4)

where

$$E_{p_k^1 p_k^2} = \frac{1}{2}(d(y^{(p_k^1)}, y^{(p_k^2)}))^2 = \frac{1}{2}\sum_{i=1}^{M}(y_i^{(p_k^1)} - y_i^{(p_k^2)})^2.$$

To minimize E^{att}, we update the weights using the steepest descent method as follows:

$$w_{ij} \leftarrow w_{ij} - \eta \frac{\partial E_{p_k^1 p_k^2}}{\partial w_{ij}}, \qquad (5)$$

for k = 1,..., $\boldsymbol{P}_k$. The thresholds are adjusted in a similar way as that for $\boldsymbol{E}^{rep}$.

The procedure for operating this algorithm is similar as the former one. We randomly pick a pair of patterns from the same class. We use these two patterns as input vectors (denoted as $\mathrm{x}^{(p_k^1)}$ and $\mathrm{x}^{(p_k^2)}$) and feed them into the network to obtain output responses (denoted as $y^{(p_k^1)}$ and $y^{(p_k^2)}$). We calculate the distances between every pair of output vectors, which are produced by patterns in the same class. We pick the pair which has the maximum distance. Then, use this pair of output vectors and their corresponding input patterns in (5) to decrease the distance.

We may employ a mixed strategy to operate the repellence force in (2) and the attraction force in (5) in a sequential mode. We randomly select two patterns from all classes. When these two patterns come from a same class, we use (5) to pull them close together; when they come from different classes we use (2) to push them far apart from each other. The network is trained until the following two conditions are satisfied: (1) the maximum distance among all the pairs of output vectors belonging to the same class is below a predetermined threshold. (2) The minimum distance among all the pairs of output vectors belonging to different classes exceeds a predetermined threshold. Otherwise, the training will continue until no more improvement in either the maximum or minimum distance can be achieved.

3. Multilayer perceptron encoder

The way to construct the encoder for the multilayer perceptron, shown in Fig. 1, is to extend this algorithm backwards to a deep bottom layer as the BP algorithm does.

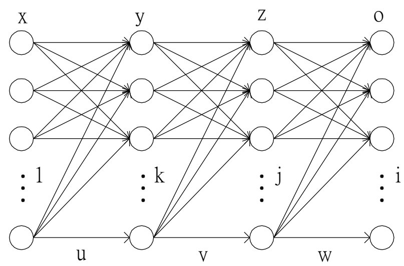

Fig. 1: A multilayer network.

The reason for doing so is that we can take advantage of the nonlinear mapping ability of a multilayer perceptron to obtain ideal representations in the output layer. We expect that a multilayer perceptron will have the potential to uniformly distribute the representations on hypercube corners and to map the fewest corners for each class with noisy patterns. The derivation is similar to that for the BP algorithm. As before, we require that the distances between the output representations of different classes must be maximized. The weights between the output layer and its connected hidden layer are adjusted by the same updating rule used in (2). All the lower hidden layers are trained backwards. The local gradient of the upper layer is propagated to the next lower layer, and their weights are adjusted accordingly. The energy function is

$$E^{rep} = -\frac{1}{2}\sum_{p_1=1}^{P}\sum_{p_2=1}^{P}(d(o^{(p_1)}, o^{(p_2)}))^2$$

$$= \sum_{p_1=1}^{P}\sum_{p_2=1}^{P} E_{p_1 p_2}$$

$$E^{rep} = -\frac{1}{2}\sum_{p_1=1}^{P}\sum_{p_2=1}^{P}\sum_{i=1}^{I}(o_i^{(p_1)} - o_i^{(p_2)})^2 \qquad (6)$$

The local gradient $\boldsymbol{\delta}_{oi}$ for the output neuron $\boldsymbol{o}_i$ is defined as

$$\delta_{o_i} = \frac{\partial E}{\partial o_i}\frac{\partial o_i}{\partial net_i},$$

where $\boldsymbol{o}_i$ is obtained as in (2). We calculate the local gradients for different input patterns p_1 and p_2. They are

$$\delta_{oi}^{(p_1)} = (o_i^{(p_1)} - o_i^{(p_2)})(\frac{1}{2}(1-(o_i^{(p_1)})^2)) \quad and$$

$$\delta_{oi}^{(p_2)} = (o_i^{(p_1)} - o_i^{(p_2)})(\frac{1}{2}(1-(o_i^{(p_2)})^2)).$$

Accordingly, the local gradients for hidden neurons are obtained as

$$\delta_{zj}^{(p_1)} = \frac{1}{2}(1-(z_j^{(p_1)})^2\sum_r \delta_{or}^{(p_1)} w_{rj},$$

$$\delta_{zj}^{(p_2)} = \frac{1}{2}(1-(z_j^{(p_2)})^2\sum_r \delta_{or}^{(p_2)} w_{rj};$$

$$\delta_{yk}^{(p_1)} = \frac{1}{2}(1-(y_k^{(p_1)})^2\sum_r \delta_{zr}^{(p_1)} v_{rk},$$

$$\delta_{yk}^{(p_2)} = \frac{1}{2}(1-(y_k^{(p_2)})^2\sum_r \delta_{zr}^{(p_2)} v_{rk}.$$

The equations listed above show that the local gradients are the weighted sums of the local gradients of their connected upper layer. Then the weights can be updated by the local gradient:

$$\Delta w_{ij} = \delta_{oi}^{(p_1)} z_j^{(p_1)} - \delta_{oi}^{(p_2)} z_j^{(p_2)},$$

$$\Delta v_{jk} = \delta_{zj}^{(p_1)} y_k^{(p_1)} - \delta_{zj}^{(p_2)} y_k^{(p_2)},$$

$$\Delta u_{kl} = \delta_{yk}^{(p_1)} x_l^{(p_1)} - \delta_{yk}^{(p_2)} x_l^{(p_2)}.$$

We may reverse the sign of $\boldsymbol{E}^{rep}$ to obtain the attraction energy. This attraction energy provides attraction forces among representations in the same class. We skip its algorithm. We operate these two kind energies for every two patterns according to their class membership.

4 Experiments and Discussions

4.1 Characters recognition

In this section we test the proposed method with experiments. The first experiment is recognition of characters. The pattern set contains 52 characters (A to Z and a to z). Each character is stored as a binary image of size of 16 pixels × 16 pixels as shown in Fig. 2. Each pattern is a vector containing one image. Each pattern is a class of its own. We construct a single-layer perceptron with 256+1 input units and 256 output neurons. Each output neuron is fully connected with all input units and a threshold unit. The training results

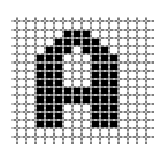

Fig. 2: A character image.

are shown in Fig. 3, Fig. 4 and Fig. 5. In Fig. 3, the bottom green line marked 'input', we plot the sorted 52 minimum distances for all 52 characters where each minimum distance is the distance between each character and its closest character among the rest 51 characters. For each output encoded representation of a single character, we calculate its Hamming distances to the rest 51 output representations and record the minimum one in the four color lines above the bottom green line. These four lines record the performances under different initial conditions and different numbers of hidden layers. As shown in Fig. 3, the minimum distances for all characters are all less than 90, the curve marked with -input-. The minimum distances of the coded representations are all greater than 100. The red curve marked with '-output 1-' is obtained by using orthogonal initial weights with one hidden layer. The green curve marked with '-output 2-' is obtained by using small random initial weights with one hidden layer. We also use the multilayer perceptron with three hidden layers as in Fig. 1 to do this experiment and plot the performances in this figure. In the multilayer perceptron, each layer has 256 neurons. The dark blue curve marked with '-output 3-' is obtained by setting the orthogonal initial weights for all layers. The light red curve marked with '-output 4-' is obtained by setting small random initial weights. From this figure, the encoded representations have larger distances than those of input patterns. It will be relatively easier to discriminate these representations in the output coding space than discriminate the image patterns in the input space by using the Hamming distance.

To see the distribution of these representations, we assume each representation evenly distributed on hypercube corners, ($2^{256}/52$), in the hidden hypercube space. The output representation of each pattern is considered as a center of these corners. Therefore, a center should be at a 222 Hamming distance to another center. The radius of the center is less than 111, because $\sum_{i=0}^{111}\binom{256}{i} > 2^{256}/52$. Thus the distance between two centers is approximately 222. This kind radius is ideal. The experiments show that we can separate the representations with a distance more than half the idea radius. To show, roughly, the sizes of these trained separation radius, we also plot the maximum Hamming distance between each encoded representation and the rest 21 representations in Fig. 4 in a similar way as those for Fig. 3. We also plot the averaged Hamming distance for each representation and the 21 rest representations in Fig. 5 in a similar way as for Fig. 3. As shown in Fig. 4, several maximum Hamming distances approach the ideal radius 222. With such well separated representations on the hypercube space, one can resolve noisy characters in the outputs of the MLP.

We may use the output representations obtained by this single layer perceptron as inputs to train the second hidden layer. Then use the output representations of the second hidden layer as inputs to train the third hidden layer. The performance of the output representations of the third hidden layer is similar to the performance curve, the '-output 4-'. In this case all layers have 256 neurons.

4.2 Image restoration

In the next experiment, we use the encoder to develop the network shown in Fig. 6 as an associative memory. There are three layers in this network, the input units, the hidden layer, and the output layer. This network is a replicator network with feedbacks. The response of the output layer will be send back to the input layer in the next iteration. There are only two layers with sigmoid function neurons. The input layer distributes signals to the hidden layer directly without any modification. The input layer and the hidden layer are used to develop highly separable internal representations for the above 52 patterns to tolerate noisy patterns. The output layer is used to index these representations to their corresponding patterns. As an associative memory, the output will evolve to a stable state gradually. This stable state is the place where we store the pattern. Given a corrupted pattern (*search argument*), one corresponding stored pattern will be recalled through the association of this corrupted pattern and a memorization mechanism.

The training algorithm of this network is divided into two stages. In the first stage, we train the weights between the input units and the hidden layer. In the second stage, we train the weights between the hidden layer and the output layer by the BP algorithm using the 52 internal representations as inputs and their corresponding patterns as the desired outputs. In this case, each layer contains 256 neurons plus one fixed unit with value -1. All neurons in a layer are fully connected to the neurons of the next upper layer. In the first stage, we use small random numbers as initial weights to start the training of the weights between the input units and the hidden layer. The results of the training are included in the former section. We then save the 52 internal representations as inputs and their corresponding patterns as the desired outputs, $\{(y^p, x^p), P=1,..,52\}$, and use them to train the weights between the hidden layer and the output layer. In the second stage, all trained weights between the input units and the hidden layer must be fixed. In this stage, we only train the weights between the hidden units and the output layer using the BP algorithm:

After training, we feed corrupted patterns to the network. The corrupted patterns are generated by randomly reversing 30% of the 256 image pixels. Successive responses of the output layer are recorded in Fig. 7. Fig. 7 shows the refined characters for the first five iterations. Most corrupted patterns will evolve to their stable patterns within five iterations. To our knowledge, this is the best performance among all existing methods.

This encoder exhausts the flexibility of all neuron's weights to accomplish widely separated and isolated new representations of all patterns in mapped space. One can develop refined representations for patterns layer after layer or train a multilayer network backwardly to obtain such representations. This encoder also accomplishes an ideal MLP kernel for the SVM for two-class problem.

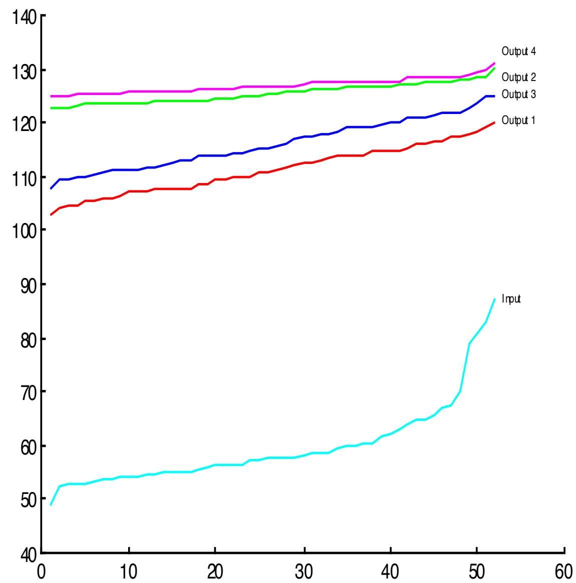

Fig. 3: Bottom green line plots the minimum Hamming distances for each of the 52 characters. Those minimum Hamming distances for coded representations are plotted above with different training parameters.

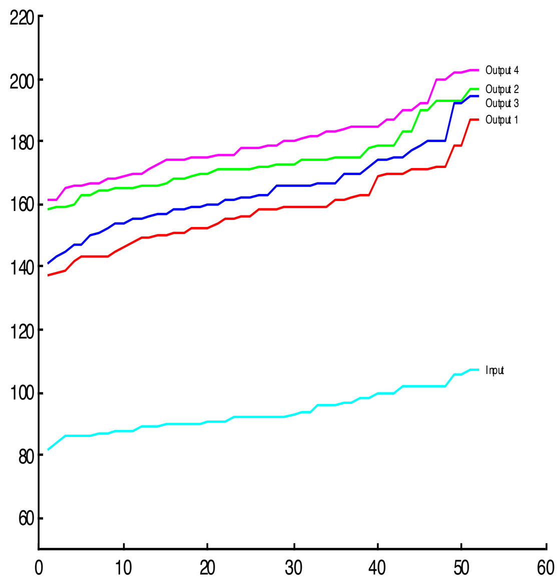

Fig. 4: The maximum distance between each encoded representation and its most remote encoded representation among the rest 51 representations.

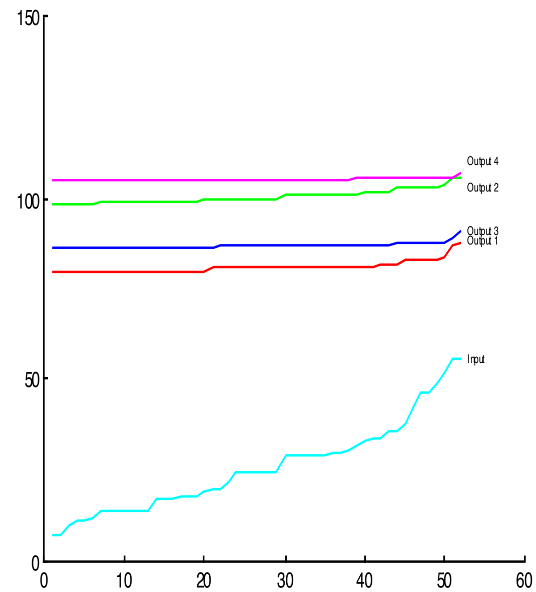

Fig. 5: The averaged Hamming distances for the 52 representations.

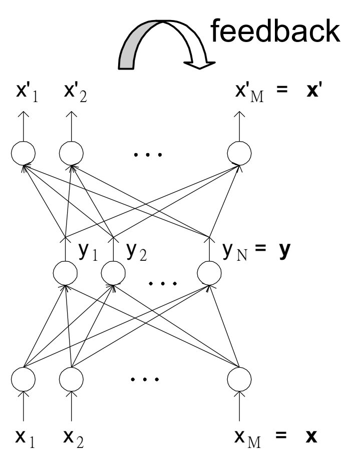

Fig. 6: The recurrent network

Fig. 7: Evolutionary recall of noisy characters. Training characters are listed in the first column. The 30% corrupted characters are in the second column. The recalled characters for the first five iterations are in the rest five columns.

Finally, we briefly discuss the E^{rep} algorithm. In the algorithm, the repellence energy E^{rep} is applied only to those nearest neighbors along the border of different classes, Fig.8. These neighbors are included in circles in the figure. These neighbors are the most sensitive patterns for discrimination and are the major patterns that cause difficult errors, local minimums, during the BP training. The algorithm suggests that one can insert a fixed perceptron right in between the nearest neighbors in each circle as possible before any BP training. These fixed perceptrons can discriminate sensitive patterns and generate faithful representations for such neighbors without training. The insertion technique is similar to those in the Bi-perceptron algorithm with half strip. Or, one can operate the SVM for nearest neighbors circle after circle, Fig.8, to get the inserted perceptron.

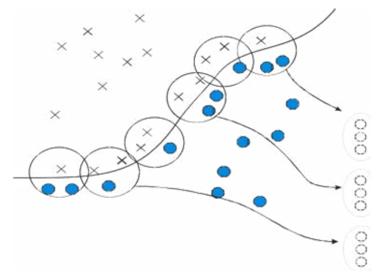

Fig. 8: Neighbors of different classes

References

[1] S. Becker, G.E. Hinton, "Self-organizing neural network that discovers surfaces in random-dot stereograms," Nature,355, 1992, pp. 161-163

[2] C. Cortes, V.N. Vapnik, "Support vector networks," Machine Learning, 20,1995, pp. 273-297

[3] C.-Y. Liou, W-J. Yu, "Initializing the weights in multilayer network with quadratic sigmoid function," International Conference on Neural Information Processing, ICONIP, pp. 1994, 1387-1392 , October 17-20, Seoul,

[4] M. Mézard, J.-P. Nadal, "Learning in feed-forward layered networks: The tiling algorithm," Journal of Physics A, 22, 1989, pp. 2191-2203

[5] W. Pedrycz, J. Waletzky, "Neural-network front ends in unsupervised learning," *IEEE Trans. on* Neural Networks 8, 1997, pp. 390-401

[6] D.E. Rumelhart, G.E. Hinton, R.J. Williams, "Learning representations by back-propagating errors," Nature (Longon), 323, 1986, pp. 533-536

[7] S. Sabour, N. Frosst , G.E. Hinton, "Dynamic Routing Between Capsules," 31st Conference on Neural Information Processing Systems. 2017, pp. 3859–3869

[8] T. J. Sejnowski, "Storing covariance with nonlinearly interacting neurons," Journal of Mathematical Biology, 4, 1977, pp. 303-321

[9] T. J. Sejnowski, C. R. Rosenberg, "NETalk: a parallel network that learns to read aloud," The Johns Hopkins University Electrical Engineering and Computer Science Tech. Report, JHU/EECS-86/01, 1986