

A Hierarchical Self-Organizing Approach for Learning the Patterns of Motion Trajectories

Weiming Hu, Dan Xie, and Tieniu Tan, *Senior Member, IEEE*

Abstract—The understanding and description of object behaviors is a hot topic in computer vision. Trajectory analysis is one of the basic problems in behavior understanding, and the learning of trajectory patterns that can be used to detect anomalies and predict object trajectories is an interesting and important problem in trajectory analysis. In this paper, we present a hierarchical self-organizing neural network model and its application to the learning of trajectory distribution patterns for event recognition. The distribution patterns of trajectories are learnt using a hierarchical self-organizing neural network. Using the learned patterns, we consider anomaly detection as well as object behavior prediction. Compared with the existing neural network structures that are used to learn patterns of trajectories, our network structure has smaller scale and faster learning speed, and is thus more effective. Experimental results using two different sets of data demonstrate the accuracy and speed of our hierarchical self-organizing neural network in learning the distribution patterns of object trajectories.

Index Terms—Hierarchical self-organizing neural network, trajectory analysis and learning, anomaly detection, behavior prediction.

I. INTRODUCTION

VISUAL surveillance has attracted much attention in computer vision due to its potential applications. In a visual surveillance system, the main problems include object detection, object classification, tracking and event recognition. In recent years, event recognition has been widely considered [1], [2]. Trajectory analysis is one of the basic problems in event understanding and is the focus of this paper. Other important issues such as object tracking are discussed elsewhere [17]–[20].

Most current visual surveillance and event recognition systems depend on known scenes, where the objects move in predefined ways [3]–[6]. These methods are not adaptable to changing environments, because for each scene one set of object behaviors should be defined, and the definition of object behaviors should be updated as object behaviors change. Furthermore, it is hard to predefine all object behaviors even when the environment does not change. It is highly desirable to establish a general approach for event recognition based on automatically generated behavior models. Johnson *et al.* [7] described a statistical model for object trajectories generated from image sequences. The movement of an object is described

by a sequence of flow vectors. Each vector consists of 4 elements that represent the position and velocity of the object in the image plane. The statistical model of object trajectories is formed with two two-layer competitive learning networks that are connected with leaky neurons. Both networks are trained using vector quantization in which only the winning neuron is excited and the other neurons are retained. Johnson *et al.* [8] generalized the model in [7] to the learning of interactions among humans, for example shaking hands. Stauffer *et al.* [9] presented a method very similar to [7] to learn patterns of activity using real-time tracking. This method involves developing a codebook of representations using an on-line vector quantization on the entire set of representations acquired by the tracker. Joint co-occurrence statistics are accumulated over the codebook by treating the set of representations in each sequence as an equivalency multiset. Finally, a hierarchical classification is performed using only the accumulated co-occurrence data. However, these methods are not applied to anomaly detection or to activity prediction. Sumpter *et al.* [10] presented a novel approach for learning long-term spatio-temporal patterns of objects in image sequences, using a neural network paradigm to predict future behavior. Owens *et al.* [11] determined whether a point on a trajectory is abnormal using the distribution of flow vectors. This method does not represent the distribution of trajectories, so neither recognizes behaviors nor predicts them. Fernyhough *et al.* [12] established the spatio-temporal region by learning the results of tracking objects in a video sequence and constructing a qualitative event model by qualitative reasoning and statistical analysis.

In this paper, we propose a hierarchical self-organizing neural network approach for learning trajectory patterns in the context of event recognition. We develop the side links of neurons to form some lines, thus, the second competitive network in [7], [10] with very slow learning speed is skipped. Corresponding to the hierarchical self-organizing neural network model, we develop two neighborhoods, namely the neuron neighborhood and the internal net neighborhood. The neurons in the both neighborhoods update their weights to different extent in the learning process. Thus, the trajectory patterns can be learned. Based on the learned patterns, abnormal events can be detected and object behaviors can be predicted. Two sets of training data from an indoor traffic scene and an outdoor campus scene respectively are used to validate the algorithms.

II. TRAJECTORY CODING

The goal of trajectory coding is to acquire the training data. The trajectory coding method in this paper is the same as that used in [7], [10] to which the following description refers.

Manuscript received January 4, 2002; revised October 15, 2002. This work was supported in part by the NSFC by Grant 60105002, the Natural Science Foundation of Beijing by Grant 4031004, the National 863 High-Tech. R&D Program of China by Grants 2002AA117010 and 2002AA142100, the LIAMA Project, and the Institute of Automation, Chinese Academy of Sciences.

The authors are with the National Laboratory of Pattern Recognition, Institute of Automation, Chinese Academy of Sciences, Beijing 100080 China (e-mail: wmlu@nlpr.ia.ac.cn; xdan@nlpr.ia.ac.cn; tnt@nlpr.ia.ac.cn).

Digital Object Identifier 10.1109/TNN.2003.820668

Given an image sequence, we obtain a series of trajectories by tracking the centroid of each object over time. (The object centroids are detected automatically by visual algorithms. Please see our previous papers [17]–[20] for more detail.) In our experiments, trajectories are sampled at a fixed rate (once every Δt frames). Let the two-dimensional (2-D) image coordinates of the centroid of an object o at the i th sampling be (x_i, y_i) . After sampling n times, we obtain a point sequence T_o that is composed of n pairs of 2-D image coordinates

$$T_o = \{(x_1, y_1), (x_2, y_2), \dots, (x_i, y_i), \dots, (x_{n-1}, y_{n-1}), (x_n, y_n)\}. \quad (1)$$

The motion direction of the object o at time i is: $D_i = \{(x_{i+1} - x_i)/D, (y_{i+1} - y_i)/D\}$, where $D = \sqrt{(x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2}$. The speed of the object o at time i is represented by the distance between the two successive points: $v_i = \sqrt{(x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2}$. We can use (dx_i, dy_i) ($dx_i = x_{i+1} - x_i, dy_i = y_{i+1} - y_i$) to represent the velocity of the object o at time i . Instead of the point sequence T_o , an object trajectory is equivalently described by a sequence of flow vectors. Each flow vector f_i represents the position of the object, and its instantaneous velocity in the image plane at time i , that is, $f_i = (x_i, y_i, dx_i, dy_i)$, $1 \leq i \leq n$. Flow vectors are transformed so that each component lies in the interval $[0, 1]$ (i.e., $x, y, dx, dy \in [0, 1]$). The relative scaling of velocity and positional components are chosen to balance their relative contribution to the measure of similarity between flow vectors. Thus an object o is, after preprocessing, represented by a set Q_o of n flow vectors: $Q_o = \{f_1, f_2, \dots, f_i, \dots, f_{n-1}, f_n\}$.

III. A HIERARCHICAL SELF-ORGANIZING NEURAL NETWORK

A. Self-Organizing Feature Map

The Kohonen self-organizing feature maps neural network [13]–[16] is composed of an input layer and an output layer. Each neuron in the input layer is linked with each neuron in the output layer by weights. It is assumed that $X \in R^M$ is the input vector, W is the matrix of weights, and $Y \in R^N$ is the matching response of the output neurons. At time t , for each output neuron j , the output is calculated by: $Y_j = W_j^T(t) \oplus X(t)$. The symbol \oplus represents an operation that can be defined in two ways: 1) dot product, i.e., $Y_j = W_j^T(t) \cdot X(t)$. In this case, the winning neuron holds the maximum of all dot products of the input and the weights of each output neuron; 2) Euclidean distance, i.e., $Y_j = \|W_j(t) - X(t)\|$. Here the winning neuron holds the minimum output.

Each output neuron has a neighborhood N_c in the output layer. The winning neuron and its neighbors in region N_c are activated to different extents, while neurons outside N_c are retained. $\alpha(t)$ is a gain coefficient in the range $(0, 1)$. Mathematically, we have the following.

$$\frac{dw_{ij}}{dt} = \begin{cases} \alpha(t)(x_i(t) - w_{ij}(t)) & \forall j \in N_c \\ 0 & \forall j \notin N_c \end{cases} \quad (2)$$

where N_c and $\alpha(t)$ decrease as t increases.

The network learns by an unsupervised self-adaptive training of the weights as each training sample is considered in turn.

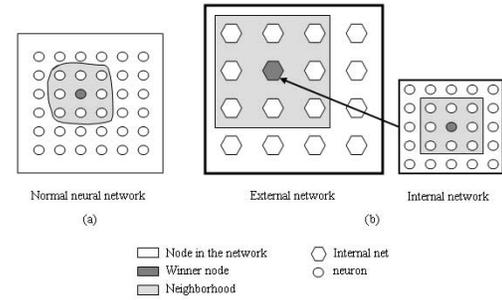


Fig. 1. Comparison between normal neural network and our hierarchical neural network.

After a number of trainings, topologically similar samples are mapped to adjacent output neurons.

B. Hierarchical Self-Organizing Neural Network

In the self-organizing feature map, neurons in the output layer are relatively independent and their relationships are defined by neighborhoods. In many cases, a set of neurons that have certain intimate relationships constitutes a group that is defined as an internal net; all neurons are partitioned into several internal nets; and all internal nets constitute an external net. The relationships between neurons in an internal net are defined by neuron neighborhoods. An internal net can be treated as a “big neuron” and the relationships between internal nets in the external net are defined by internal net neighborhoods. As a consequence, we can form a hierarchical self-organizing neural network model composed of several internal nets and an external net. Fig. 1(a) shows a normal neural network with 6×6 neurons and a neighborhood with 3×3 neurons. Fig. 1(b) shows a hierarchical neural network with 4×4 internal nets, one of which consists of 5×5 neurons, and an internal net neighborhood with 3×3 internal nets. The hierarchical neural network is used to model complicated relationships between neurons.

In the hierarchical self-organizing neural network model, the neurons in an internal net are trained with the self-organizing feature map learning method, i.e., the winning neuron and the ones in its neighborhood are activated, while neurons outside the neighborhood are retained. Internal nets in the external net are also trained with the self-organizing feature map learning method, i.e., the internal nets in the internal net neighborhood are activated to different extents, while internal nets outside the internal net neighborhood are retained. The hierarchical self-organizing learning process can be further understood from the learning of trajectory distribution patterns as described in the next section.

It should be mentioned that there are other hierarchical self-organizing neural networks [21], [22], but the neural network structures and learning process of these networks are different from ours. In [21], there are self-organizing maps at two levels: the state map and the dynamics maps. The dynamics maps are associated with each node of the state map and used to predict the next state of the state map. The learning process consists of two phases: first the training of the state map and next the training of the dynamics maps. In [22], the hierarchical self-organizing map is composed of two self-organizing maps.

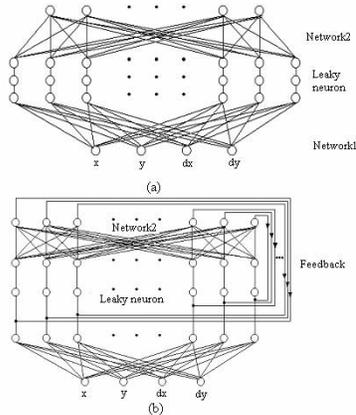


Fig. 2. Existing network structures. (a) Neural network structure used in [7]. (b) Neural network structure used in [10].

First, for each input vector, the best matching unit is chosen from the first map and its index is input to the second map. Second, the best matching unit from the second map is chosen and its index is the output of the hierarchical network. The existing hierarchical self-organizing map methods have a multilayer structure and two training phases (we refer to [21], [22] for more details). In our hierarchical self-organizing structure, all neurons are partitioned into several internal nets; the internal nets and the external net are both in the same layer; and there is no two stage training phase, unlike existing hierarchical self-organizing maps.

IV. LEARNING ALGORITHM

The motivation of learning is to model the probability density functions of trajectories. In this section we first compare our neural network structure with the existing ones, and then introduce the learning algorithm.

A. New Network Structure

As mentioned in Section 1, previous work [7]–[10] models the probability density functions of motion trajectories via two two-layers competitive learning networks that are connected with leaky neurons. The network structures are shown in Fig. 2(a) and (b). In Fig. 2(a), the first network acquires the distribution model of flow vectors. A (x, y, dx, dy) corresponding to a flow vector is used as an input vector. The number of output neurons equals the number of flow vectors. The output of the first neural network is inputted to the second network. The second network builds the distribution of trajectories. The number of its input neurons equals that of output neurons in the first neural network. Its output neurons correspond to trajectories. The learning speed of the second network is much slower than that of the first one. In Fig. 2(b), feedback is introduced to the second competitive network in Fig. 2(a) giving a more efficient prediction of object behavior. This idea is very original. However, the number of input and output neurons in the second network remains to be the number of flow vectors, so the efficiency of learning decreases inevitably as the size of the network increases. Furthermore, this neural network structure cannot be used to detect unusual behaviors.

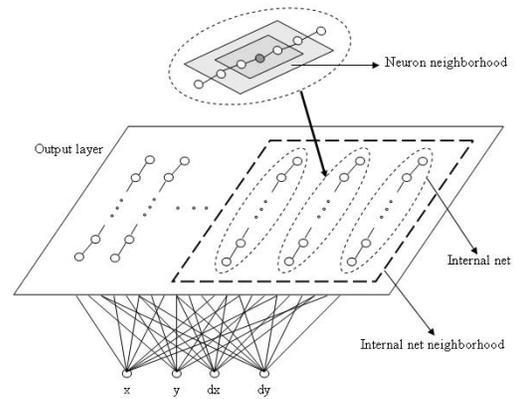


Fig. 3. Network structure used in this paper.

We introduce the side links between neurons to the first neural network in the existing neural network structures as shown in Fig. 3. The output neurons are linked to l lines, i.e., the side links between the neurons are built (this is different from existing methods). Each line corresponds to a trajectory and forms an internal net. The relationship between all internal nets forms the external net. Each neuron corresponds to a flow vector f , whose components, in turn, correspond to the weights of the neuron. The neural network is learned from a sequence of corresponding movements among neurons, and gradually organizes toward an optimal solution in which the distribution of output neurons is consistent with the distribution of flow vectors in training samples, and the distribution of the neuron lines (internal nets) is consistent with the distribution of training trajectories. In this way we skip over the second neural network with a large scale and very slow learning speed, which is used to build the distribution patterns of trajectories, in the existing neural network structures. Our network structure is much simpler than the existing ones, but it can build the same distribution patterns of trajectories as the others. Therefore, it is more effective. This is confirmed by the experimental results. We explain the neuron neighborhood and the internal net neighborhood shown in Fig. 3 in Sections IV-C and IV-D.

B. Initialization

At the initialization phase, the weights of neurons are set to random values. The number of internal nets used to describe the distribution patterns of trajectories is essentially arbitrary. The more internal nets are used, the greater the accuracy of the mode and in turn the longer the learning time. The number of internal nets should be much less than that of trajectory samples. Before the learning process, all internal nets have the same number of neurons as the maximal number of vectors in each sample trajectory. After the learning process, by the adjustment of the internal nets, trajectories of different length are modeled (see Section IV-F).

The trajectory samples should be normalized to the same length. Assuming that one sample trajectory has n sampling points, where (x_n, y_n) represents the coordinates of the last point in the trajectory, and each trajectory should be normalized to N sampling points, $N-n$ flow vectors all represented by $(x_n, y_n, 0, 0)$ are padded to the sample trajectory so as to get a

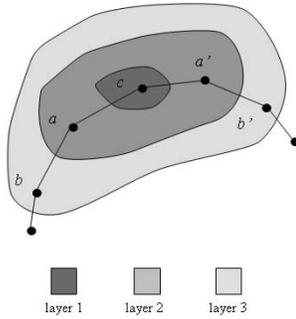


Fig. 4. Neuron neighborhood.

vector made up of N sampling points. In this way, all the sample trajectories are normalized to the same length.

C. Definition of Neuron Neighborhood

In our algorithm, the winning neuron c and the neurons that directly or indirectly connect with it form a neuron neighborhood, as shown in Fig. 4. A neighborhood can be represented by a series of layers. As shown in Fig. 4, neuron c makes up layer 1; neurons a and a' which directly connect with c make up layer 2; neurons b and b' which directly connect with neuron a or neuron a' which is in layer 2 make up layer 3; ...; neurons which directly connect with neurons in layer $k-1$ make up layer k . The neighborhood of neuron c at time t is represented by $N_c(t)$ which is composed of neurons within layer $k(t)$ relative to the neuron c . So $k(t)$ which decreases as t increases determines the size of neighborhood. The layer function $L(k)$ represents the level of activation between neurons in layer k and neuron c : $L(k) = 1/Z^{k-1}$ ($Z \geq 2$).

D. Definition of Internal Net Neighborhood

As the initial weights are set to random values, if we only adjust the weights of the neurons in the internal net that contains the winning neuron, some neurons are idle (i.e., they cannot be activated), and some neurons are busy. This directly affects the learning results. The internal net neighborhood is used to tackle this problem, as shown in Fig. 3. We define the internal net that best matches the trajectory sample that includes the input flow vector as the center of the internal net neighborhood. All the internal nets that have matching relationships with the input vector are included within the internal net neighborhood. For an input flow vector belonging to the trajectory Q_o , the matching relationship between an internal net and Q_o is determined by the number of points in the internal net that best match the points in Q_o . The more neurons in the internal net that match the points in Q_o , the higher the matching degree between the internal net and Q_o . The matching degree also determines the sensitivity by

which the neurons in the different internal nets in the current internal net neighborhood adjust their weights, i.e., the higher the matching degree, the more intensively the neurons activate, and vice versa.

E. Training of Weights

Based on the two neighborhoods defined earlier, we can apply the hierarchical self-organizing learning method to learning the distribution patterns of trajectories. When inputting a flow vector $x(t)$ assumed to belong to trajectory Q_o , the learning process includes the following two steps. First we select the internal net in the output layer that best matches Q_o as follows:

```
int num[l];
```

```
for (k = 1; k <= Len(Q_o); k++) {
```

```
    For each flow vector  $f_k \in Q_o$ , find neuron  $c$  that best matches  $f_k$ . The Euclidean distance between neuron  $c$  and flow vector  $f_k$  is minimum:
```

$$\sqrt{\sum_{i=1}^4 (x_i(f_k) - w_{ic'})^2} = \min_j \left(\sqrt{\sum_{i=1}^4 (x_i(f_k) - w_{ij})^2} \right). \quad (3)$$

```
    If neuron  $c$  belongs to internal  $L_k$ , then  $\text{num}[L_k] = \text{num}[L_k] + 1;$ 
```

```
}
```

```
If  $\text{num}[m] = \max_k(\text{num}[L_k])$  is satisfied,
```

```
then the internal net best matching trajectory  $Q_o$  is line  $m$ .
```

The second step is the adjustment of weights. We define the internal net m as the center of the current internal net neighborhood, and each internal net i satisfying $\text{num}[i] > 0$ is included in the internal net neighborhood. For each internal net i in the current internal net neighborhood, we find the neuron c on internal net i that corresponds to $x(t)$ in order. The weights of neuron c and the neurons in the neighborhood of neuron c (N_c) are updated as shown in (4) at the bottom of the page.

Function $\alpha(t)$ is a scalar-valued function of t and decreases as t increases. The function $\alpha(t)$ can be defined by $\alpha(t) = Ae^{-t/\tau}$, where τ is a positive constant and $0 < A \leq 1$, or by $\alpha(t) = 0.9(1 - t/n)$, where n is the total number of iterations.

Function $\gamma_i(t)$ represents the sensitivity of the neurons in internal net i in the current internal net neighborhood: $\gamma_i(t) = \text{num}[L_i] / \max_k(\text{num}[L_k])$.

Neighborhood N_c shrinks gradually as t increases until there is only one neuron in N_c . The size of the neighborhood $K(t)$ is represented with the maximum topological distance between the winning neuron and neurons in the neighborhood. $K(t)$ can decrease linearly with the increase in t : $K(t+1) = K(t) - 1$. The value $K(0)$ should not be too large, because there is some incorrect matching at the beginning and the limit to $K(0)$ can reduce the negative effect of such mismatching.

$$\begin{cases} w_{1j}(t+1) = w_{1j}(t) + \alpha(t) \times \gamma_i(t) \times L(|j-c|) \times (x_1(t) - w_{1j}(t)) \\ w_{2j}(t+1) = w_{2j}(t) + \alpha(t) \times \gamma_i(t) \times L(|j-c|) \times (x_2(t) - w_{2j}(t)) \\ w_{3j}(t+1) = w_{3j}(t) + \alpha(t) \times \gamma_i(t) \times L(|j-c|) \times (x_3(t) - w_{3j}(t)) \\ w_{4j}(t+1) = w_{4j}(t) + \alpha(t) \times \gamma_i(t) \times L(|j-c|) \times (x_4(t) - w_{4j}(t)) \end{cases}. \quad (4)$$



Fig. 5. Tracking of toy vehicles in model scene.

After the flow vectors belonging to the trajectory Q_o are used to train the network, the flow vectors in the next trajectory are inputted. One loop of learning is finished when all the flow vectors in the training set have been used once. The learning process terminates if the predefined number of iterations is achieved, or the change of neuron weights in this loop of learning is less than a predefined value ε

$$\max_{i,j} \{|w_{ij}(t+1) - w_{ij}(t)|\} < \varepsilon \quad (5)$$

otherwise another loop of learning is implemented. Equation (5) is called the stability condition.

F. Adjusting the Length of Each Internal Net

For the sake of the convenience of learning, the internal nets each have the same number of neurons and the trajectory samples are normalized to the same length. After learning is finished, each internal net is adjusted to the original length that corresponds to the original trajectory samples that still were not normalized to the same length. The adjustment method is introduced in the following.

For each internal net j , we find all trajectory samples $S = \{S_1, S_2, \dots, S_K\}$ which best match it. An array $\text{number}[N]$ is used to record the number of padded points for the samples to be normalized. For each sample S_k in the sample set S , if N_k points are padded during the length normalization, $\text{number}[N_k] = \text{number}[N_k] + 1$. If $n = \max_{N_k} \{\text{number}[N_k]\}$, i.e., in the sample set S , most samples need be padded by n points, then we truncate n points at the latter part of internal net j . Thus trajectories of different lengths are modeled.

V. ANOMALY DETECTION AND BEHAVIOR PREDICTION

After the learning procedure and based on the weights of neurons, we can judge whether one event is abnormal according to the trajectory produced by the observed event, and can predict the future trajectory along which the object will move according to the observed partial trajectory.

A. Anomaly Detection

After we represent a trajectory T_o by a set Q_o of flow vectors, for each flow vector $f_i = (x_i, y_i, dx_i, dy_i)$ in Q_o , we find the output neuron that best matches it. If the distance between the flow vector and the neuron is higher than a threshold q , the point in the trajectory represented by the flow vector is flagged as unusual. If the number of unusual points in a trajectory is higher than a predefined value, the event represented by the trajectory is marked as abnormal. The threshold q is decided as follows: for each flow vector f_i in the samples, we find the neuron which best matches it and calculate the Euclidean distance p_i between

the neuron and f_i . The threshold is 50% of the maximum $p_i: q = 1/2 \max_i p_i$.

Johnson *et al.* [7] can only judge whether the whole trajectory is abnormal, but cannot detect the parts of trajectory where the abnormality occurs. Our algorithm can not only judge whether the whole trajectory is abnormal, but also detect whether part of the trajectory is abnormal and point out the abnormal part.

B. Prediction

If we only have a partial trajectory T_o , for each flow vector f_i in trajectory T_o , we find the neuron that matches it best, and record the matched line by an array $\text{number}[l]$. If the flow vector f_i matches a neuron belonging to line j , $\text{number}[j] = \text{number}[j] + 1$. Knowing the array $\text{number}[l]$, we can calculate the probabilities of the partial trajectory T_o matching neuron lines. It is assumed that N is the number of flow vectors in T_o and P_j is the probability of T_o matching line j , and then $P_j = (\text{number}[j]/N) \times 100\%$. P_j represents the probability of the object moving along the trajectory represented by line j . Let k be the index for which $\text{number}[k]$ is the maximum of the array $\text{number}[l]$. The line k is the most likely trajectory along which the object will move.

VI. PROCEDURE AND COMPLEXITY

The procedure of the algorithm is described in the following:

- Step 1) Build the set of training data $X = \{(x_i, y_i, dx_i, dy_i)\}$ by coding the sample trajectories; Due to the reason described in Section II, We normalize x_i, y_i, dx_i, dy_i to the interval $[0, 1]$.
- Step 2) Initialization. Randomize the initial weights of the output neurons, and normalize the input samples to the same length, etc.
- Step 3) Train the weights W of the neural network using the hierarchical self-organizing learning method.
- Step 4) Adjust the lengths of all internal nets.
- Step 5) Apply the distribution patterns learned to anomaly detection and behavior prediction

The running time is mainly spent on learning the distribution patterns of trajectories. The quality of the learning result depends on the number of learning steps. The more learning steps, the better the results. Generally, the number of iterations is $O(n)$ (n is the number of output neurons) and the number of weights that should be updated upon every input is $O(n)$, so the time complexity of the self-adaptive process is $O(n^2)$.

VII. EXPERIMENTAL RESULTS

We implemented all algorithms using Visual C++ 6.0 on the Windows 2000 platform. The tracking of moving objects is the



Fig. 6. Tracking of pedestrian in a campus scene.

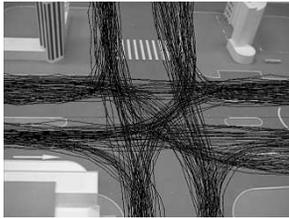


Fig. 7. Two hundred and one sample trajectories in model scene.



Fig. 8. Two hundred sixty-eight sample trajectories in outdoor campus scene.

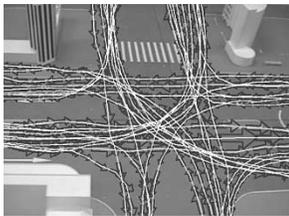


Fig. 9. Result learned with our method in model scene. After 102 iterations (14.15 s).

basis of our work. In most cases, the tracked objects are vehicles or pedestrians. Moving objects are tracked in the image plane to obtain a series of trajectories and each point on a trajectory indicates the centroid of the tracked object. The centroids of the object at different time are connected to form the trajectories. The following two groups of training data are used in this paper.

- 1) The first group of training data is acquired by tracking the moving toy vehicles controlled by radio in a traffic model scene, as shown in Fig. 5. The moving vehicle tracked is labeled with a white '+' sign, whose center is at the centroid of the object.
- 2) The second group of training data is acquired by tracking moving objects in a real outdoor campus such as pedestrians, bikes and cars. An example of pedestrian tracking is shown in Fig. 6. (Figs. 5 and 6 illustrate how we acquire sample trajectories.)

With continuous tracking, we acquired two sets of training trajectories, as shown in Figs. 7 and 8. There are 201 trajectories in the model scene as shown in Fig. 7, and 268 trajectories in the outdoor campus scene as shown in Fig. 8.

Fig. 9 shows the distribution of the neurons and the distribution of the neuron lines in the output layer after the learning finished. A white line represents an internal network in our neural network, and corresponds to a trajectory. Each neuron in the output layer is displayed as an arrow, whose centroid represents an object position corresponding to the components (x, y) , and whose size and direction represent respectively an object's motion velocity corresponding to the components (dx, dy) . There are 58 lines in total. The network stability condition is $\varepsilon = 0.001$ corresponding to Formula 5. To reach the stability condition, 102 iterations and 14.15 s running time are needed (all the data of running time in this paper are calculated on a Pentium3-933 computer with 256M RAM). As shown in Fig. 9, there are no oscillations and the learned patterns of trajectories are consistent with the sample trajectories, so the results can be treated as acceptable.

Fig. 10 shows the distribution of neurons in the output layer in the campus scene at the phase of initialization and after 5, 10, 20, 50, and 162 iterations. The number of lines is 48. The network stability is the same as that used in the first example corresponding to Fig. 9. As shown in Fig. 10, with the number of iterations increasing the patterns of lines gradually approximate to the sample trajectories. The comparison between these lines and the sample trajectories shows the learned results to be plausible.

In the following, we compare our method with that used in [7]. In [7], the movement of objects is described using the positions and velocities of the objects in the image plane as the same as ours. The neural network structure used in [7] has been introduced in Section IV-A. Its scale is large and thus affects the learning speed and the learning results. In [7], vector quantization is used to train the neural network. Different from the Kohonen self-organizing feature map, vector quantization only updates the weight of the winning neuron, keeping the weights of other neurons unchanged. Fig. 11 is the learning result of the method used in [7] for the samples from the model scene when the network stability condition is the same as Fig. 9. To reach the stability condition, 797 iterations and 39.88 s running time are needed. Fig. 12 shows the learning process of the method used in [7] for the samples from the campus scene when the network stability condition is the same as Fig. 10. Comparing Fig. 11 with Figs. 9, and 12 with Fig. 10, we find that the method used in [7] needs more iterations and more running time than ours to reach the same stability condition. In addition, it produces local oscillations in the leaned patterns of trajectories. These oscillations lead to unacceptable results while our results are acceptable as aforementioned. Table I demonstrates the number of iterations and the running time required by the two training algorithms for the model scene with different stability conditions. For each

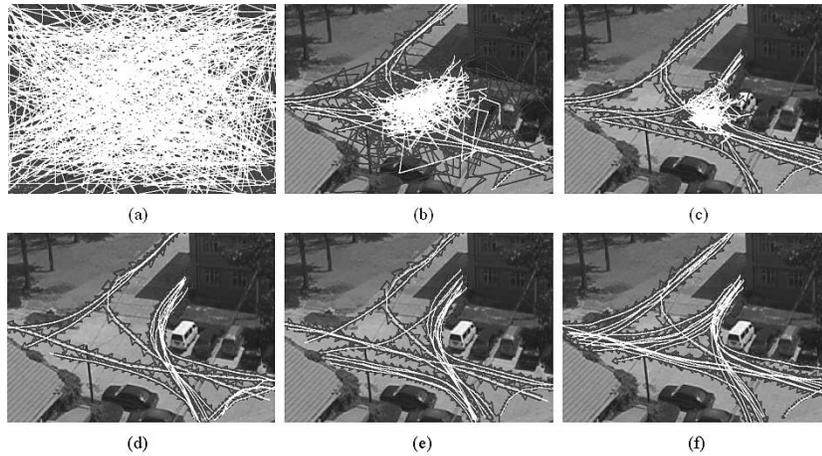


Fig. 10. Learning process with our method in campus scene. (a) Random initial weights. (b) After 5 iterations (0.63 s). (c) After 10 iterations (1.32 s). (d) After 20 iterations (2.65 s). (e) After 50 iterations (7.20 s). (f) After 162 iterations (23.39 s).

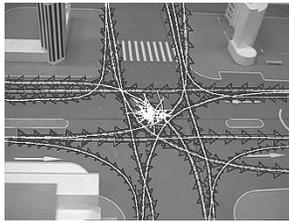


Fig. 11. Result learned with vector quantization (presented in [7]) in model scene. After 797 iterations (39.88 s).

stability condition, the experiment was performed three times with different initial weights. From Table I, we can see that our method needs fewer iterations and less running time to reach the same stability condition than that used in [7]. The reason our method is more effective than that use in [7] rests with the following two aspects.

- We skip over the second competitive network in [7] whose scale is big, resulting in a slow learning speed.
- In [7], vector quantization is used to train the network. In experiments we find that when vector quantization is used to learn the distribution patterns of trajectories, most neurons are not excited at the early stage of training and shift toward the center of samples just according to the weight sensitivity determined in the learning process. This slows down the speed of the network convergence and greatly affects the learning accuracy.

Fig. 13 is an example of anomaly detection in the model scene. The car entered the scene from the left then turned right. The trajectory of the car is shown as a series of arrowheads. Abnormal points are marked with white “x” signs at the center of arrowheads. As shown in (a), (b), and (c), points too close to the center of the road are detected. When the car made the turn, it moved within the proper region. However, when the car began to run down, it ran into the driveway for the reverse direction. Several abnormal points are, therefore, desirably detected as shown in (f), (g), (h). As mentioned in Section V-A, if there is more than the certain number of unusual points (e.g., three

points) in a trajectory, the event represented by the trajectory can be marked as abnormal.

Fig. 14 is an example of anomaly detection in the campus scene. In (a) and (b), a pedestrian walked within a normal region. In (c), the pedestrian entered a parking lot, and then he entered the region (a grassplot) where admission is forbidden [as shown in (d), (e)]. After he left the region, he walked in a wrong direction [as shown in (f), (g)]. These are correctly marked as abnormal. In (h), he moved within the proper route and no anomalies were marked.

Fig. 15 shows an example of prediction in the model scene. The car entered the scene from the left and then turned left. In (a), there are three predicted trajectories for the car. In (b), the probabilities of the car running along these three trajectories are changed respectively. In (c), the rightmost trajectory is deleted because the probability of the car running along it is very small. (When the probability of an object moving along a trajectory is less than 15%, the trajectory and its corresponding probability are not shown in Figs. 15 and 16.) In (d), the tendency of the car to turn left is very clear. The middle trajectory is deleted. Another similar example of behavior prediction in the outdoor campus scene is demonstrated in Fig. 16.

In the above text, the hierarchical self-organizing neural network method is compared with the method used in [7] in the context of learning trajectory patterns, and then the results of anomaly detection and behavior prediction are demonstrated. Experimental results show that the hierarchical self-organizing neural network model can effectively learn the distribution patterns of motion trajectories, and anomaly detection and behavior prediction can both be achieved. The results of trajectory patterns learning, anomaly detection and behavior prediction are consistent with one’s visual judgement. This demonstrates the acceptable accuracy of the algorithms in learning patterns of trajectories, as well as detecting anomalies and predicting object behaviors.

VIII. CONCLUSION

In this paper, we argue that event models can be established automatically by learning rather than predefined manually, and

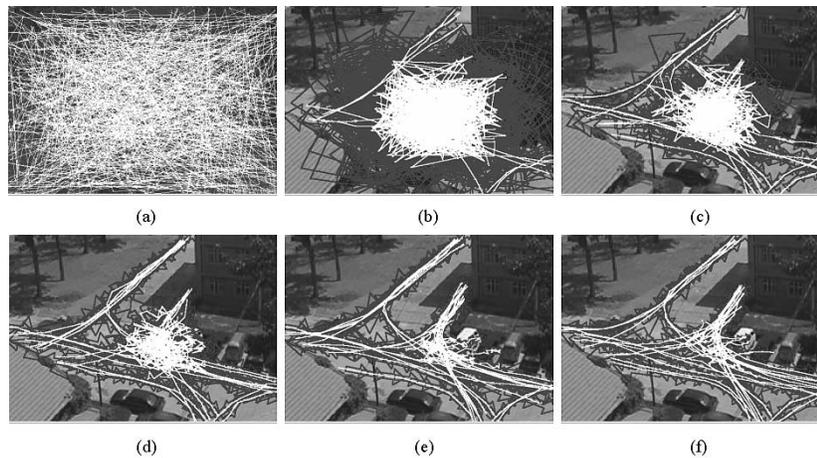


Fig. 12. Learning process with vector quantization (presented in [7]) in outdoor campus scene. (a) Random initial weights. (b) After 50 iterations (3.81 s). (c) After 100 iterations (7.61 s). (d) After 300 iterations (22.86 s). (e) After 500 iterations (34.89 s). (f) After 893 iterations (62.19 s).

TABLE I
COMPARISON OF THE REQUIRED NUMBER OF ITERATIONS

Stability condition \mathcal{E} in Formula (5)	Vector quantization		Hierarchical self-organizing neural network	
	Iterations	Computational cost	Iterations	Computational cost
0.0001	924	47.39s	147	19.50s
	817	41.91s	111	14.79s
	883	45.31s	139	18.46s
0.001	797	39.88s	106	14.15s
	822	42.17s	127	16.09s
	910	46.69s	129	17.12s
0.01	779	38.94s	94	12.61s
	667	34.20s	93	12.44s
	784	39.19s	101	13.51s

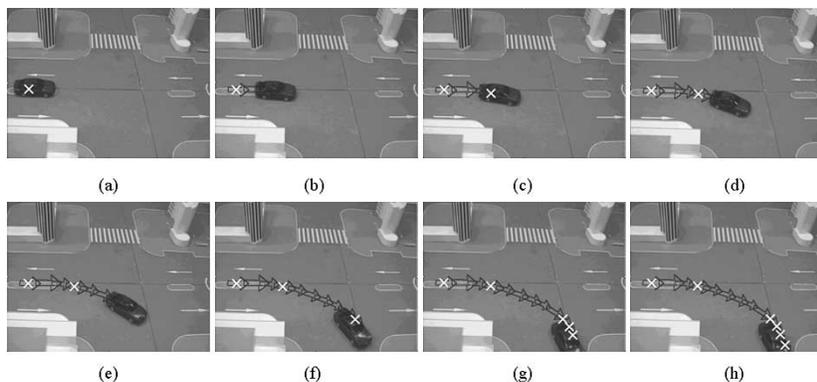


Fig. 13. Example of anomaly detection in the model scene.

focus on learning the patterns of trajectories using a self-organizing approach. The main contributions and results of this paper are as follows:

- We propose a new neural network structure for learning the patterns of trajectories. Our network structure is much simpler and more effective than the existing ones.
- Based on the new network structure, we present a hierarchical self-organizing neural network and its application to learning the distribution patterns of trajectories.

- We use the learned patterns of trajectories to detect anomalies and predict object behaviors.
- Experiments on image sequences taken in a traffic model scene and an outdoor campus scene show that our method is more effective than that used in [7], as our method needs fewer iterations and less running time to reach the same stability condition, and produces more acceptable results.

Future work will involve more robust event detection and prediction based on three-dimensional (3-D) object tracking, for

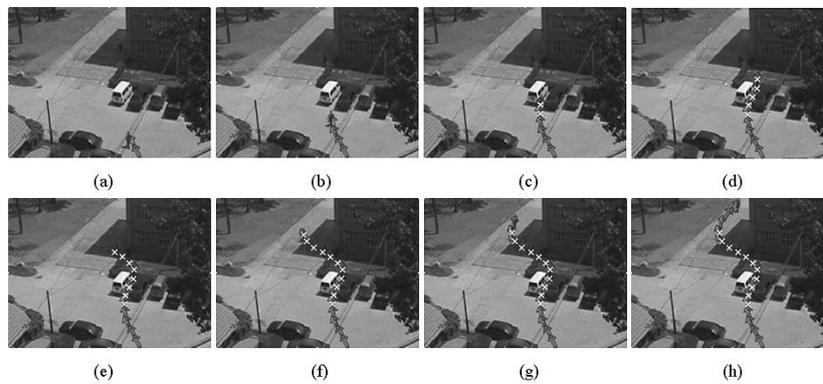


Fig. 14. Example of anomaly detection in the campus scene.

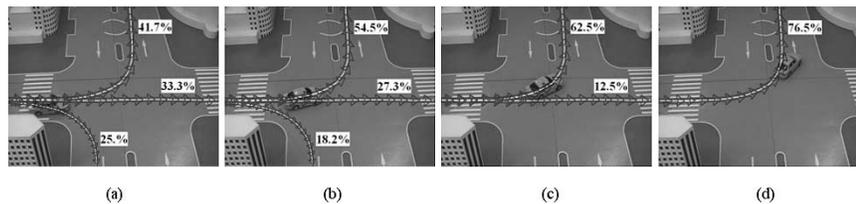


Fig. 15. Example of prediction in indoor model scene.

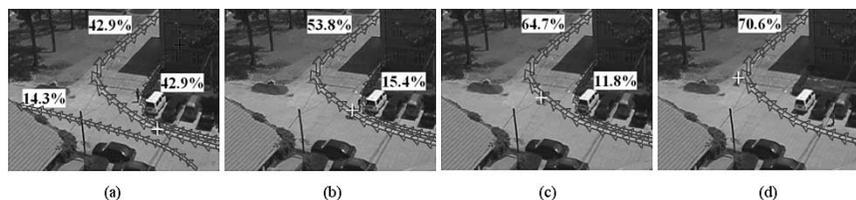


Fig. 16. Example of prediction in the campus scene.

example, by using 3-D vehicle models, so that accidents such as car crash may be predicted accurately.

ACKNOWLEDGMENT

This work is partly supported by NSFC (GrantNo. 60520120099) and Natural Science Foundation of Beijing (Grant No. 4041004).

REFERENCES

[1] T. Collins, A. J. Lipton, and T. Kanade, "Introduction to the special section on video surveillance," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 22, pp. 745–746, 2000.
 [2] R. J. Howarth and H. Buxton, "Conceptual descriptions from monitoring and watching image sequences," *Image and Vision Computing*, vol. 18, no. 9, pp. 105–135, 2000.
 [3] R. J. Howarth and B. Hilary, "An analogical representation of space and time," *Image and Vision Computing*, vol. 10, no. 7, pp. 467–478, 1992.
 [4] E. Andre, G. Herzog, and T. Rist, "On the simultaneous interpretation of real world image sequences and their natural language description: The System Soccer," in *Proc. ECAI-88*, Munich, 1988, pp. 449–454.
 [5] K. Schaefer, M. Haag, W. Theilmann, and H. Nagel, "Integration of image sequence evaluation and fuzzy metric temporal logic programming," in *KI-97: Advances in Artificial Intelligence, Lecture Notes in Computer Science, 1303*, C. Habel, G. Brewka, and B. Nebel, Eds. New York: Springer, 1997, pp. 301–312.
 [6] M. Brand and V. Kettner, "Discovery and segmentation of activities in video," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 22, pp. 844–851, 2000.

[7] N. Johnson and D. Hogg, "Learning the distribution of object trajectories for event recognition," *Image and Vision Computing*, vol. 14, no. 8, pp. 609–615, 1996.
 [8] N. Johnson, A. Galata, and D. Hogg, "The acquisition and use of interaction behavior models," in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. Silver Spring, MD, 1998, pp. 866–871.
 [9] C. Stauffer and W. E. L. Grimson, "Learning patterns of activity using real-time tracking," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 22, pp. 747–757, 2000.
 [10] N. Sumpter and A. Bulpitt, "Learning spatio-temporal patterns for predicting object behavior," *Image and Vision Computing*, vol. 18, no. 9, pp. 697–704, 2000.
 [11] J. Owens and A. Hunter, "Application of the self-organizing map to trajectory classification," *Proc. IEEE Workshop on Visual Surveillance*, pp. 77–83, 2000.
 [12] J. Fernyhough, A. G. Cohn, and D. C. Hogg, "Constructing qualitative event models automatically from video input," *Image and Vision Computing*, vol. 18, no. 9, pp. 81–103, 2000.
 [13] T. Kohonen, *Self-Organizing Maps*: Springer-Verlag, 1995.
 [14] W. M. Hu, J. H. Xu, X. L. Yan, and Z. J. He, "Partitioning on MCM using a new neural network model," *Science in China Series E*, vol. 42, no. 3, pp. 312–320, 1999.
 [15] W. M. Hu, C. C. Li, Y. S. Zhu, and X. L. Yan, "Neural network approach for timing, power dissipation and wire connection driven placement," *Chinese J. Semiconductors*, vol. 20, no. 9, pp. 797–803, 1999.
 [16] M. C. Su and H. T. Chang, "Fast self-organizing feature map algorithm," *IEEE Trans. Neural Networks*, vol. 11, pp. 721–733, 2000.
 [17] H. Yang, J. G. Lou, H. Z. Sun, W. M. Hu, and T. N. Tan, "Efficient and robust vehicle localization," in *IEEE Int. Conf. Image Processing*, 2001, pp. 355–358.
 [18] J. G. Lou, H. Yang, W. M. Hu, and T. N. Tan, "An illumination invariant change detection algorithm," in *Asian Conf. Computer Vision*, 2002, pp. 13–18.

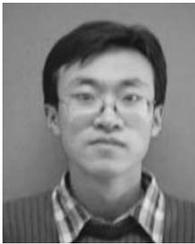
- [19] J. G. Lou, H. Yang, W. M. Hu, and T. Tan, "Visual vehicle tracking using an improved EKF," in *Asian Conf. Computer Vision*, 2002, pp. 296–301.
- [20] Y. Tian, T. N. Tan, and H. Z. Sun, "A novel robust algorithm for real-time object tracking," *Chinese J. Automation*, vol. 28, no. 5, pp. 851–853, 2002.
- [21] O. Simula, E. Alhoniemi, J. Hollmen, and J. Vesanto, "Monitoring and modeling of complex processes using hierarchical self-organizing maps," in *IEEE Int. Symp. Circuits Syst. (ISCAS'96)*, 1996, pp. 73–76, volume Supplement.
- [22] M. Goktepe, N. Yalabik, and V. Atalay, "Unsupervised segmentation of gray level Markov model textures with hierarchical self-organizing maps," in *Int. Conf. Pattern Recog.*, 1996, pp. 90–94.



Weiming Hu received the Ph.D. degree from the Department of Computer Science and Engineering, Zhejiang University.

From April 1998 to March 2000, he was a Postdoctoral Research Fellow with the Institute of Computer Science and Technology, Founder Research and Design Center, Peking University. Since April 1998, he has been with the National Laboratory of Pattern Recognition, Institute of Automation, Chinese Academy of Sciences, as an Associate Professor. His research interests are in

visual surveillance and monitoring of dynamic scenes, neural networks, and filtering of Internet objectionable images. He has published more than 40 papers on national and international journals, and international conferences.



Dan Xie received the Bachelor's degree in automatic control from Beijing University of Aeronautics and Astronautics (BUAA), China, in 2001.

He is currently pursuing the Master's degree at BUAA, where he is majoring in computer graphics and virtual reality. His current research interests include pattern recognition, machine learning, and neural networks.



Tieniu Tan (M'92–SM'97) received the B.Sc. degree in electronic engineering from Xi'an Jiaotong University, China, in 1984 and the M.Sc., D.I.C., and Ph.D. degrees in electronic engineering from Imperial College of Science, Technology and Medicine, London, UK, in 1986, 1986, and 1989, respectively.

He joined the Computational Vision Group, Department of Computer Science, The University of Reading, England, in October 1989, where he worked as Research Fellow, Senior Research Fellow, and Lecturer. In January 1998, he returned

to China to join the National Laboratory of Pattern Recognition, the Institute of Automation of the Chinese Academy of Sciences, Beijing. He is currently Professor and Director of the National Laboratory of Pattern Recognition as well as President of the Institute of Automation. He has published widely on image processing, computer vision and pattern recognition. His current research interests include speech and image processing, machine and computer vision, pattern recognition, multimedia, and robotics.

Dr. Tan serves as referee for many major national and international journals and conferences. He is an Associate Editor of *Pattern Recognition* and of the *IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE*, as the Asia Editor of *Image and Vision Computing*. He was an elected member of the Executive Committee of the British Machine Vision Association and Society for Pattern Recognition (1996–1997) and is a founding co-chair of the IEEE International Workshop on Visual Surveillance.