Behavior Modeling by Neural Networks

Lambert Spaanenburg[1], Mona Akbarniai Tehrani[1,⋆], Richard Kleihorst[2], and Peter B.L. Meijer[2]

[1] Lund University, Dept. of Electrical and Information technology, P.O. Box 118, 22100 Lund, Sweden
lambert@eit.lth.se, sx06ma1@student.lth.se
[2] NXP Semiconductors, Research, High Tech Campus, 5656 AE Eindhoven, The Netherlands
richard.kleihorst@vito.be, Peter.b.l.meijer@nxp.com

Abstract. Modeling of human and animal behavior is of interest for a number of diagnostic purposes. Convolutional neural networks offer a constructive approach allowing learning on a limited number of examples. Chaotic tendencies make that learning is not always successful. The paper looks into a number of applications to find the reason for this anomaly and identifies the need for behavioral references to provide determinism in the diagnostic model.

Keywords: Behavioral modeling, convolutional neural network, collision avoidance, 3-tier architecture, Sinai billiard system.

1 Introduction

Neural networks have received much interest for image understanding and intelligent control. Some work has been done in categorizing social behavior of humans and animals, largely from a biological point of view to understand and model how nature works. Of late, homeland security issues kindle renewed interest. Many public places have been equipped with cameras, but the mere amount makes on-line monitoring by humans impossible. This has created the interest to automatically detect and diagnose suspicious behavior.

Smart vision sensors for automated surveillance of public places can be based on Commercially-Off-The-Shelf (COTS) processors, but specialized image processors are required for massively crunching pixels into features while achieving real-time performance at a low power budget. This poses additional restrictions on the implementation of the neural networks. Fully connected, monolithic neural networks have a limited capacity when implemented as a single digital ASIC, while they tend to give cache problems on COTS processors and usually do not agree with the fixed-point number representation popular for the cheaper

⋆ This paper is published with the support of Image Processing and Interpolation (IPI) departemnt of Gent University, Gent, Belgium.

C. Alippi et al. (Eds.): ICANN 2009, Part I, LNCS 5768, pp. 439–448, 2009.
© Springer-Verlag Berlin Heidelberg 2009

embedded platforms. This pleads for the use of heterogeneous platforms such as WiCA [1], where a conventional 8051 is combined with the IC3D ASIC, a 320-core pixel cruncher. As monolithic neural networks are ill-famed for their learning problems, hierarchical structures seem advisable, leaving the question: how to build and distribute this hierarchy on the platform.

This sets the case for the convolutional neural network, a layered hierarchy of mostly-neural modules. The paper is therefore composed as follows. First we introduce the modular hierarchy of neural networks with an illustrative application. In section 3 we discuss automotive collision avoidance and in section 4 we treat the Sinai billiard system. Finally we discuss our findings and draw some conclusions.

2 The Modular Hierarchy

The core problem in developing neural networks is collecting, validating and presenting the examples. Often it is quoted that more than 90% of the project time has been devoted to getting and preparing the data set to learn from [2]. This is the more surprising, as usually a large amount of ground knowledge exists such as basic physical laws that can be incorporated in several ways.

Firstly the networks may be initialized to approximate "ideal" behaviour. The clean network has weight values that are randomly selected small values around zero. This is meant to create no built-in preferences, so that anything can be learnt. Larger and non-random weight values will introduce preferences, as desired, but it is not always transparent how they can be calculated.

Secondly, the problem can be described by more abstract examples. To facilitate this, the abstract notion must be founded on reality by known principles, the so-called ground truth. Although the relation is well-defined, the use of fixed knowledge can aggravate the learning problem [3]. Often it is cast into a neural network so its parameters can be slightly adapted to the circumstances under which the measurements are made.

For large networks, a monolithic solution tends to be hard (if not impossible) to train [4]. This is caused by conflicting data in the training set or by data that becomes conflicting due to the training itinerary. A solution is to compose the network from modules in the form of sub-networks that can be individually initialized or trained. A fortunate side effect is the structuring and reduction of the example set. Sometimes the composition can be derived from the problem at hand [5], but the convolutional network provides a generic framework as a hierarchy of layers with parallel operating sub-networks.

The layered hierarchy is based on the combined operation of many small fully-connected feed-forward neural networks. The underlying idea is a 3-tier architecture, as popularized in many fields of telecommunication (Figure 1) with the following meaning:

– The lower (*foundation*) tier expresses the basic technological ingredients. This may introduce facts of common knowledge to the neural system, abstract fundamental physical parameters to the digital realm, or unify different graphic

packages into a single programming interface. In image understanding, we find here the low-level pixel operations for pre-processing, built from initialized sub-networks.

- The middle (*processing*) tier contains the operational functions. It provides the transformations and operations to support a domain of applications in some related technological fields. This may offer a set of classes for the modelling in the envisaged domain and/or a set of algorithms to perform numerical support. In image understanding, we find here sub-networks individually trained for a choice of medium-level blob operations.
- The final (*application*) tier provides the interface to the application at hand. It personalizes the domain to provide a direct support to the user. The functionality is expressed in terms of the processing functions. As a consequence, any changes in the application will not induce a major effort as long as they can still be expressed in the available functions. In image understanding, we find here the high-level feature operations trained for the overall problem.

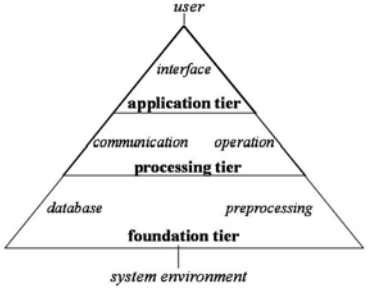

Fig. 1. The 3-tier concept

The introduction of 3 layers aims to decouple platform restrictions and hardware dependencies from the user application. In the software arena, the 3-tier architecture is directly related to concepts of re-use and maintenance; for hardware, the tiers image the integration stages between components and product. For the benefits harvested in applying the multi-tier architecture in neural systems, see [6].

This method has been first applied at full length for the detection of surface defects on steel from a rolling mill [7]. Here, the defects appear as peaks in a flat area. By their reliance on domain rather than application properties, the lower tier can be built from one single network type. Basic physical laws on light reflection are turned into a set of artificial data. Samples of light reflections in all possible gradients are used, varying around the target value characteristic for the specific sub-network. The outputs are the characteristic values corresponding to the applied physical laws but in a manner that makes the sub-networks act as weak classifiers. As the upper networks combine these characteristic values to make the final decisions, it is required that the data to train the lower tier cover the entire problem space (Figure 2).

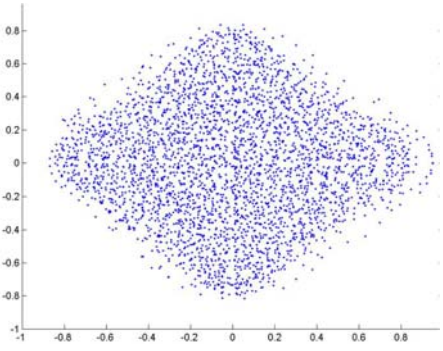

Fig. 2. Plot of target values for the lower networks. The training data represent the whole problem space.

The middle tier serves to provide parameter invariance. Rotation and size-invariance is achieved by arranging the outputs from the lower tier into so-called *peeling-layers* [7]. Width invariance occurs as a large peak and will fill all the three peeling-layers but a small peak will fill only the inner layer. We use the directions of the vectors that result from the lower layer and calculate three properties that reflect continuity in assembling these vectors into potential peaks. Additionally a threshold is applied on the results to suppress any peak detection of a structure that does not adequately reflect a peak. The three output values from the middle networks are used as input for the upper network. The 3rd tier network makes the final decision of whether the larger image is centred over a peak.

In this application, the neural system applies only to single images. It supports the detection of and measurements on defects by intelligently matching the expectations of light reflections to the actual observations. We call this 'structural stitching' in contrast to the 'situational stitching' to enforce continuity on the assembly that we discuss in the next sections.

3 Collision Avoidance

In the early 90's the European automotive industry started their first collaborative research program, called Prometheus. The next example comes from the PRO-CHIP project within that program. In the original set-up a large collection of data on a driving car was made. This collection is by itself not usable as it describes mostly just driving along the road without changes in speed and/or direction. Data editing seems necessary to eliminate catastrophic forgetting in adaptive networks, or prevent implicitly giving disproportionally high weight to simple behaviours in non-adaptive networks. For example, driving straight-ahead for a long time tends to destroy all the knowledge about taking a curve. This editing on the set can be a long and cumbersome process, as a lot of tests are necessary to check on the sensitivity of the results to the partial removal of training data.

A better approach is to define a closed set of typical driving situations, from which during learning a random selection with small random variations is made (Figure 3). Different driver characteristics can be obtained by judiciously balancing the number of specific examples in the training set. Such behaviour has been tested in a simulated environment with a variety of uncontrollable disturbances, such as side-wind while turning on a slippery road [8].

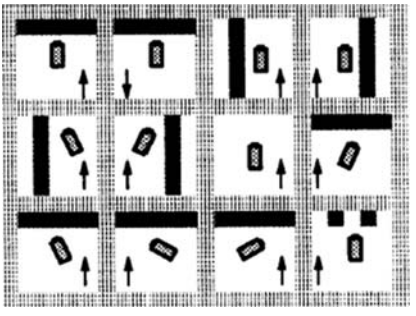

Fig. 3. Basic driving situations (not shown is the 13th situation, where no blockade is near)

A neural network is a non-linear system. Such a system may show chaotic behaviour because of sensitivity for initial conditions. In the driver model this will lead to different behaviour for every different starting position of the car. It is interesting to observe that this is true on the detailed level but on the more abstract level of reasoning as introduced by the driving examples in Figure 3, it has only occurred when the driver had not to take a curve or carefully avoid obstacles, such as the side of the road. Why is that? The answer is that the curves and blockades put the driver a straitjacket on. There are far fewer ways to leave the curve unharmed than to enter it. In other words, the curves serve as points of reference for path convergence such that with every curve accuracy gets largely restored. This becomes visible in car behaviour when comparing a route with an occasional curve with a route with many curves. For instance, after a U-turn all cars are almost at the same trajectory. The same reasoning applies to obstacle avoidance. This has two important consequences. First of all, it seems that trajectories are a meaningful signature of behaviour. The same reasoning applies to obstacle avoidance. This has two important consequences. First of all, it seems that trajectories are a meaningful signature of behaviour. Secondly, it seems that behaviour funnels acting as points of reference can restrict chaotic systems. In the experiments, both the simulation as the limited precision implementation has shown the same repeatable and deterministic trajectories. Figure 4 shows a typical testing ground; further demonstration of continuous correct driving was delivered to the 1991 PRO-CHIP meeting in Darmstadt.

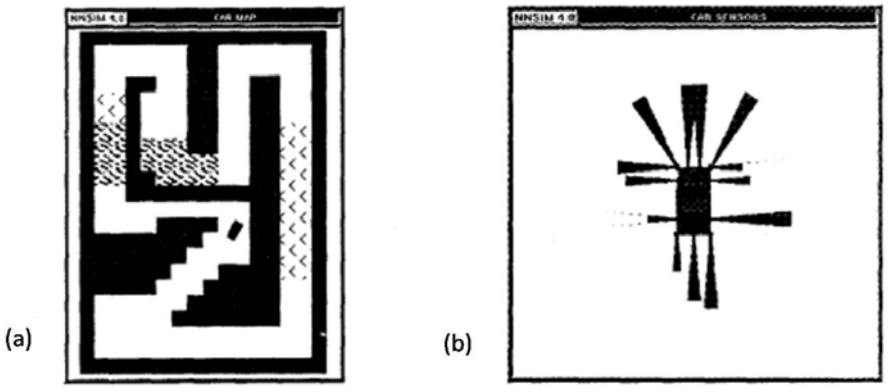

Fig. 4. (a) Screen showing a driving environment with wind (arrows) and ice (shaded roads) and (b) screen showing the actual position, direction und range of the thirteen distance sensors on the car

4 A Billiard System

Based on the aforementioned experience we now simulate a so-called "Sinai billiard table" on an embedded computer vision system with limited dynamic range in its number formats. The Sinai billiard system was introduced by Yakov G. Sinai and shows long-term unpredictability in a non-linear dynamic system with unbounded accuracy [9].

In this system a (simulated) billiard ball is rolling on a square table in straight lines and changes its direction when it hits the walls, corners or the circle in the middle of the table according to the usual laws (angle of incidence equals angle of reflection).

Having the position of the ball, its speed and direction and the knowledge of its environment (the distance to the closest obstacle), the motion pattern can be learnt in different situations separately as the ball faces different obstacles. The input flow vector to the selected neural network (NN) consists of the ball

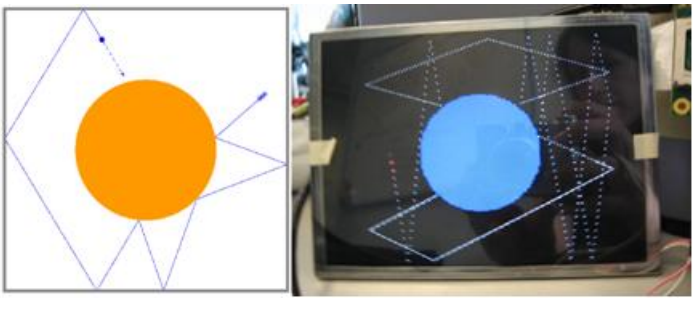

Fig. 5. Sinai billiard ball simulator

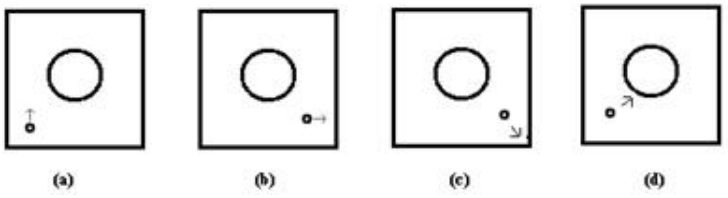

Fig. 6. Different motion patterns of the ball

coordinates in two subsequent frames *f*={*x(t-2), x(t-1), y(t-2), y(t-1)*}. In addition, coordinates of the ball in the next frame *T*={*x(t), y(t)*} are given to the networks as target outputs.

In the first case, a multilayer feed-forward neural network (FFNN) learns the linear movement of the ball (Figure 6a), which can be expressed by the formulas:

$$x[t] = 2 \times x[t-1] - x[t-2] \text{and} y[t] = 2 \times y[t-1] - y[t-2].$$

A FFNN with a 4-2-2 topology is selected and trained offline using supervised learning with the training data set prepared using a MatLab model of the Sinai billiard ball simulator. It is typical for a convolutional network that the local problem admits a simple construction (here two linear neurons would suffice), but a more generic set-up achieves an easier integration within the overall system. After training in NEUREKA [10], the network has a good generalization results and acceptable error rate (Table 1).

Table 1. Training and testing results for linear movement pattern

Train dataset size	Training error	Test dataset size	Nr. epoch	Test error	Activation function
200	0.08%	200	3000 iterations	0.09%	Sigmoid Function

For the next condition, a FFNN with similar topology is trained to learn the ball motion pattern after collision with the wall and corners (Figure 6b and c).

$$m[t] = 2 \times m\,[t-1] - m\,[t-2]\,. \tag{1}$$

$$n[t] = n[t-2] \tag{2}$$

Here $m = y$ and $n = x$ when the ball collides with vertical walls and $m = x$ and $n = y$ when ball collide with horizontal walls. Error rate in both training phase and test phase is small and the networks learn the motion pattern successfully (Table 2).

Finally, the motion pattern of the ball after collision with the circle in the middle of the table is modelled by another FFNN, with 4-6-4-2 topology. Here the ball follows the following nonlinear pattern:

$$Dist = \sqrt{(Xc - x[t-1])^2 + (Yc - y[t-1])^2} \tag{3}$$

$$nx = (x[t-1] - Xc)/Dist; \quad ny = (Yc - y[t-1])/Dist \tag{4}$$

Table 2. Training and testing results for border collisions

Train dataset size	Training error	Test dataset size	Nr. epoch	Test error	Activation function
200	0.1%	200	4000 iterations	0.09%	Sigmoid Function

$$dp = -nx \times dx[t-1] + ny \times dy[t-1] \tag{5}$$

$$dx[t] = 2 \times dp \times nx + dx[t-1]; \quad dy[t] = -2 \times dp \times ny + dy[t-1] \tag{6}$$

$$x[t] = x[t-1] + dx[t]; \quad y[t] = y[t-1] + dy[t] \tag{7}$$

All the above networks are trained and tested successfully with an error rate of around 1% (Table 3). This larger error is caused by training this network as a single structure instead of as an assembly of 2 networks, which it actually is. Therefore further improvement is easily attainable.

Table 3. Training and testing results for circle wall collisions

Train dataset size	Training error	Test dataset size	Nr. epoch	Test error	Activation function
50	0.16%	50	11000 iterations	1.4%	Sigmoid Function

5 Discussion

In the next step the networks created above are transferred and implemented on the WiCA platform. Putting everything on a single resource, i.e. the 8051 on the WiCA platform has two drawbacks [11]. Firstly, the execution of the neural networks is now based on fixed-point values and therefore precision will become an issue. Secondly, the 8051 is slow and the results only become available once per 2 frames while running the system at 30 fps with VGA resolution. However, the accelerated pixel crunching through the IC3D helps to reduce complexity by representing a blob (like a billiard ball) by only its Centre-of-Gravity coordinates.

During the testing phase the input values of the networks are calculated by parallel pixel processing on the IC3D and coordinate transfer to the 8051. As the position of the ball is known relative to the obstacles defined in the simulator, one of the above NNs is selected in each frame for predicting the next position of the ball, and input values are forwarded to that NN. The results are then sent back to the IC3D for further analysis and live monitoring.

In Figure 7 the results of applying different NNs are shown. White dots on the screen illustrate the mismatch between actual ball position and predicted position by NN (abnormal motion behaviour). Figure 7a represents the situation where only one NN is used to predict linear motion of the ball. In Figure 7b also the second NN is applied which predicts the position of the ball after collision with the border walls. In Figure 7c the third NN is added, predicting position of the ball after collision with the circle in the middle of the screen. In some places faulty mismatches occur due to modelling errors and also because of random

Fig. 7. Results of applying different FFNN to simulator

errors, which are enforced occasionally in the simulator to overcome numerical problems caused by the use of fixed-point values with limited dynamic range.

Observation of the performance and results of the above system lets us identify the problems and challenges facing NN implementations in an embedded platform [1].

- Offline learning and model generation outside the embedded platform as implied by the convolutional network architecture allows optimized modelling for the platform.
- Depending on the word length of the input values and the size of the selected neural network, the outputs may become available with different delays due to different processing times, making it more difficult to have uniform results for monitoring and further analysis.
- The collisions with the wall do not provide a convergence reference for the subsequent behaviour and therefore chaotic movement (drift) sets in. This becomes even more apparent for situations where the round ball collides with a round object that amplifies small differences in angles of incidence.

Fig. 8. A set of observed walking trajectories

All by all, we have a movement that is easy to predict locally, but hard over longer periods, and that will remain so even when more computational precision is available. Even though a normal surveillance system monitoring human movement will have less chaotic moving elements, a convergence reference will be needed. We find this also in modelling larger-scale walking patterns. Figure 8 demonstrates that over some distance the walking patterns can be adequately modelled by carefully stitching the micro-level motion details together while raising the accuracy of the collisions. After some time, remaining chaotic effects of the collisions in the movement have died out and the pattern can be used again for prediction over larger distances.

References

1. Kleihorst, R., Schueler, B., Danilin, A., Heijligers, M.: Smart camera mote with high performance vision system. In: Digest DSC 2006 (2006)
2. Schuermann, B.: Applications and Perspectives of Artificial Neural Networks, VDI Berichte, vol. 1526, pp. 1–14 (2000)
3. Jansen, W.J., Diepenhorst, M., Nijhuis, J.A.G., Spaanenburg, L.: Assembling engineering knowledge in a modular multilayer Perceptron neural network. In: Digest ICNN 1997, pp. 232–237 (1997)
4. Auda, G., Kamel, M.: Modular Neural Networks: A Survey. J. of Neural Systems 9, 129–151 (1999)
5. terBrugge, M.H., Nijhuis, J.A.G., Spaanenburg, L., Stevens, J.H.: CNN Applications in toll driving. J. of VLSI Signal Processing 23(2/3), 465–477 (1999)
6. Nguyen, C.T.: Method and system for converting code to executable code using neural networks implemented in a very large scale integration (VLSI) integrated circuit, U.S. Patent 6,578,020 (2003)
7. Grunditz, C., Walder, M., Spaanenburg, L.: Constructing a neural system for surface inspection. In: Proc. IJCNN, vol. III, pp. 1881–1886 (2004)
8. Nijhuis, J., Hofflinger, B., Neusser, S., Siggelkow, A., Spaanenburg, L.: A VLSI implementation of a neural car collision avoidance controller. In: Proc. IJCNN, vol. 1, pp. 493–499 (1991)
9. Sinai, Y.G.: Dynamical Systems with Elastic Reflections. Russian Math. Surveys 25, 137–191 (1970)
10. Meijer, P.B.L.: Neural Network Applications in Device and Subcircuit Modeling for Circuit Simulation, PhD thesis, Eindhoven University (1996)
11. Tehrani, M.A.: Abnormal motion detection and behaviour prediction, M.Sc. thesis, Lund University (2008)