Simulation of Network Machines

Thorsten Pöschel - Humboldt-Universität zu Berlin, GDR.

1. Introduction

The simulation of neuronal systems has got two aims:

First to get information about the mechanism of human information processing systems. That means to find out the relations between the microstructure, which is well investigated by biology and physiology, and the macrostructure investigated by cognitive psychology. Both sciences are highly developed; the working mechanism of a single or some neurons as well as many properties about the behaviour of whole nervous systems are well understood. The cognitive psychology tries to develop models, which conclude from the reactions of test persons to information processing mechanisms at a lower level. By means of psychological experiments it seems possible to verify or falsify macroscopical neuronal models /7/.

From the microstructure one tries to approach more complex systems by investigating Neural Networks. The investigation of two special types of so called Network Machines is the subject of the presented contribution.

The System Theory also tries to conclude from the behaviour of simple elements to the properties of more complex systems and also in reverse direction. However in difference to neuronal systems with about $10^{10}$ simple elements the objects of System Theore are much smaller.

The second aim of the investigation of neuronal nets concerns the development of artificial intelligent systems, e.g., learning by example presentation machines, which change an inherent state according to the presented task, the required result and the error the machine has made, by means of a special feedback mechanism.

2. Network Machines

A network machine as we will understand it in this paper consists of a large number $N$ of simple uniform processing elements. Each of them is characterized by its activation $E_i$ and its output $a_i$, which is a analytic or probabilistic function of the activation $f(E_i)$:

$$a_i = f(E_i)$$  \hspace{1cm} (1)

The processing elements are connected by weighted links in a special manner, which is characterized by the type of network machine. These links $w_{ij}$ transfer the output $a_i$ of an element to the input of another. Some of the elements serve as interface to the environment. The information, which should by processed (the problem) expressed is binary numbers or real numbers out of a limited interval, is clamped onto the input elements of the net. Then the other elements get an activation transferred by the links and change their output according to the output function (1). In this way also the states of the output elements vary. From these numbers the output of the network can be obtained. Now the deviation of the output from the required by a teacher result has to be determined and the weights must be varied in such a manner, that the deviation will reduce /8/.

In the following sections we will discuss two special types of Network Machines and present some of the results of computer simulations we have done.

3. The Boltzmann Machine

The Boltzmann Machine consists of bistable elements, which can acquire the values 0 or 1. Each of them is designated by its bias $b_i$. The weights of the links are real numbers $w_{ij}$, which connects the output of the $j$-th element with an input of the $i$-th one. The weight are symmetrically, e.g. $w_{ij} = w_{ji}$. All working elements are connected with each other. The output of an element $a_i$ here is a stochastic function of its activation. With probability

$$P(E_i) = \frac{1}{1 + e^{-\frac{E_i}{T}}}$$  \hspace{1cm} (2)

the state of the element $i$ is set to 1 otherwise it becomes 0. This formula is well known from the Metropolis Algorithm, used to calculate partition functions. In this algorithm $T$ has the meaning of the thermodynamic temperature. Assuming that $E$ is the internal energy of the Boltzmann Machine

$$E = \sum_{i=1}^{N} E_i = \sum_{i=1}^{N} \sum_{j=1}^{N} w_{ij} a_i a_j$$

$$E_i = \sum_{j=1}^{N} w_{ij} a_j - b_i = E(a_i=0) - E(a_i=1)$$

(3)

It is shown, that obeying these formulae processing elements converge to a probability distribution

$$P(a) = C \cdot e^{-\frac{E(a)}{T}}$$  \hspace{1cm} (4)

where $a$ is the statistical ensemble of the outputs of the working elements.

To describe the learning rule of the Boltzmann Machine we define a measure for the deviation of the networks result from the desired one, called information gain:

$$G = \sum_{a} P(a) \ln \frac{P(a)}{P_{\text{ref}}(a)}$$  \hspace{1cm} (5)

$P$ is the probability, that the configuration $a$ of the output elements occurs when the network is running freely and $P$ is the desired by a teacher probability for this configuration. That means $P$ is obtained by running the Boltzmann Machine according to the Metropolis algorithm, where the input elements and the output elements are clamped to the inputs given by a teacher and all others are freely running. Corresponding $P_{\text{ref}}$ is the probability which the Boltzmann Machine finds itselfen when the network is connected to the teaching input and all other elements are running the Metropolis algorithm. $G$ has got the property:

128
\[ G = 0 \text{, for } P = P \]
\[ G > 0 \text{, otherwise} \]  

(6)

\[ G = 0 \] is the aim of the learning procedure, e.g., in this case the network comes to the same equilibrium state, independent of the clamped or unclamped output elements. For this reason we may use \( G \) as measure of the deviation of the networks output to the teaching one. That means, we will change the weights according to:

\[ \Delta w_i = -\epsilon \left( \frac{\partial G}{\partial w_i} \right) \]  

(7)

and \( w_i = w_i + \Delta w_i \)

where \( \left( \frac{\partial G}{\partial w_i} \right) \) is an averaged over some different input patterns value. As result of a long calculation one finds /L:

\[ \frac{\partial G}{\partial w_i} = -\frac{1}{T} (P_i - P'_i) \]  

(8)

where \( P'_i \) is the probability that \( a \) and \( b \) are 1 for freely running outputs; \( P_i \) is the corresponding probability for the case, that the outputs are clamped; \( \epsilon \) is a constant /L./

4. Performance of the Boltzmann Machine

In the sections 4.1. and 4.2. we present computer simulations of the Boltzmann Machine we performed with a personal computer ATARI 260 ST (4.1.) and with a SIM 52 computer (4.2.). It turns out that even such simple problems as our examples require a large amount of computer power. More complex examples, as in the case of the Error Propagation Machine have not been done for this reason.

4.1. The 4-2-4 Encoder

The 4-2-4 Encoder is a standard example to test the performance of learning networks proposed by /L./ The 4-2-4 Encoder consists of 4 input elements, 2 inner elements and 4 output elements (fig.1). There are allowed different input configurations and also for a fixed task 4 teaching outputs, two examples of tasks are shown in fig. 2.

![Fig. 1](https://via.placeholder.com/150)

![Fig. 2](https://via.placeholder.com/150)

The difficulty is to press the 6 bit information of the input elements through the bottleneck of 2 inner ones. For this reason of course the input elements are not directly connected with the output ones, otherwise the task turns out to be trivial. After about 80 representations of every input pattern the network found the correct output with a reliability of 100%.

In fig. 2 the normalized values of the inner energy as defined in (3) of the completely taught with example 2 in fig. 2 network is drawn for the 4 possible input configurations and the corresponding 5 configurations of the inner and output elements which correspond to the lowest inner energy values. As expected it is obvious that the configuration with the lowest inner energy corresponds to the taught output.

![Fig. 3](https://via.placeholder.com/150)

4.2. Simple Pattern Recognition

In our second example we want to discuss a simple problem of pattern recognition. The machine, consisting out of 25 input, 20 inner and 20 output elements was taught to recognize 20 different 3 x 3 patterns independent of their location within a 5 x 5 raster as shown in fig. 4 /L./

![Fig. 4](https://via.placeholder.com/150)

Also this very computationally consuming task was solved and after learning the machine recognized the different patterns with a rather high reliability (fig. 5).

![Fig. 5](https://via.placeholder.com/150)
5. Error Propagation

The second example for Network Machines we want to discuss is the Error Propagation Machine. In difference to the Boltzmann Machine it consists of elements whose output is a real number out of the interval (0,1). The processing elements are arranged in layers, the first layer consists of input elements, the last one of output elements and all others of inner elements. In our calculations we used only 1 layer of inner elements; the incorporation of more layers appears to be not reasonable in our examples.

The weights of the links that are directed from top to down are signified by a real number, \( w_i \in \mathbb{R} \) (fig. 6).

![Diagram of a Network Machine](image)

fig. 7

The activation \( B \) of an element is given by:

\[
B = \sum w_i z_i + \theta
\]

(9),

where the summation runs over all elements \( z_i \) of the layer above. The output of an element is an analytic function of its activation:

\[
w = f(B) = \frac{1}{1 + e^{-B}}
\]

(10)

The working mechanism acts as follows: First the input of a learning example given by a teacher is clamped to the input elements of the network. Then the output values of the inner elements from the 2nd layer are calculated according to (9) and (10). The output values of the next layer, either an inner one or the output layer is calculated in the same manner and so on, until the output layer is reached. Then we determine the deviation \( \text{ERROR} \) of the network's output \( \{s_i\} \) from the teachers correct answer \( \{x_i\} \) as Euclid distance (11).

\[
\text{ERROR} = \sum_i (s_i - x_i)^2
\]

(11)

The summation runs over all output elements. Then, very similar to the Boltzmann Machine, the weights of the links are varied in a manner, that the \( \text{ERROR} \) will reduce. Again \( \frac{\partial \text{ERROR}}{\partial w_i} \) is the averaged derivative over some different output patterns. In our Simulations we averaged by means of an exponentially decaying filter \( \mu \in \mathbb{R} \):

\[
\Delta w_i = \mu \left( \frac{\partial \text{ERROR}}{\partial w_i} \right)
\]

(12)

\[
w_i = w_i + \Delta w_i
\]

6. Performance of the Error Propagation Machine

In the next two sections we present some Examples of the performance of the Error Propagation Machine. This Simulations have been performed with an IBM XT (6.1, 6.2.) and an ATARI PC (6.3.1).

6.1. The Enhanced (4×N)-2-4 Encoder

The Enhanced Encoder consists of 4×N input, 2 inner and 4 output elements. The problem the network has to solve is similar to the encoder problem in 4.1, with the additional difficulty that only 4 of the 4×N input elements carry an information. These 4 elements are hidden within the input layer. The teaching inputs of the others are randomly chosen, e.g. the additional task consists in finding out which input elements are significant for the output by means of examples. As in 4.1, the net solves this task very quickly.

Analyzing the number of connection values \( w_i \) out of an given interval (fig. 8) it turns out that only very few weights have got a meaningful value. The peak at the value around zero originate from the weights of the links from the not significant input elements to the inner ones. These inputs should have no influence to the output of the network and for that reason the weights connected with these elements are about zero.

![Diagram of an Enhanced (4×N)-2-4 Encoder](image)

fig. 8

6.2 Comparison of Binary Numbers

The task consists in finding out whether the binary number of the first 8 input elements greater than the binary number of the last 8 ones. The used network has got 16 input, a single inner and 1 output element. (fig.9)

![Diagram of a Comparison of Binary Numbers](image)

fig. 9
If the output element $x$ had a value $> 0.5$ the output was interpreted to be 1, otherwise 0. If the network didn't capture any information about the problem yet, in the beginning of the learning procedure it will decide correct with a reliability of about 50%, if it captured the meaning of the most significant bits of both binary numbers it decides with 75% reliability. When the net knows the meaning of both next significant bits it decides with 82.5% correct and so on. After 300–400 teaching procedures the reliability was about 100%, e.g. the net learned to classify the input elements with reference to their importance.

6.3. Pattern Recognition

One of the most discussed problem in modern informatics is pattern recognition. However it is evident that other smaller and highly developed techniques attain a higher reliability yet. In this section we want to demonstrate that networks are also capable to recognize simple patterns as capital hand drawn letters.

For this example we used a network of 150 input, 20 inner and 26 output elements. The input elements were arranged in a 10 x 15 matrix. The machine was taught with 12 alphabets by different authors. Four of this alphabets are shown in fig. 10.

```
A    B    C    D    E    F    G    H    I    J
K    L    M    N    O    P    Q    R    S    T
U    V    W    X    Y    Z       Z    Y    W    V    U
```

It is obvious that the shape of the curve is different, but the peak is nearly at the same point at an Hamming distance of about 85.

7. Evolution Algorithm

Analyzing the distribution of the $w_i$ values of the former pattern recognition task we found that the majority of the elements of the taught matrix are around zero (fig. 14). That means they have nearly no influence on the performance of the network. We set all the $w_i$ whose values lie in the hatched area to zero (fig. 15), e.g. we disconnected the corresponding working elements. It turns out that the reliability of the network remains the same.
This fact induces the idea to combine the Error Propagation Machine with an evolution algorithm /5,6,9/. This algorithm runs for a three layered network as follows:

1. Connect every inner element with a certain number of input and output elements, initialize them with random numbers.
2. Perform a number of learning cycles.
3. Disconnected every inner element links with a small w-value.
4. Connect the inner elements with other input and output elements using the free links originating from the former step.

Applying this algorithm to the pattern recognition task from 3.3. we did not get a significant improvement yet. It took us nearly the same computational time to achieve the same reliability as in the fully connected net. Until now the advantage consists in the fact that we used only 706 weighted links instead of 3566, e.g. we use only about 20 % of the memory to store the matrix w.

8. Conclusions

Two different types of learning networks were investigated as special cases of self-organization of networks (for more details see /8/). We have seen that networks with the same structure are able to solve different tasks only by presenting learning examples, e.g. not using any kind of a program. The question what kinds of tasks network models are suitable for is unsolved yet. We think, that the Cognitive Psychology is able to supply a lot of experiences and experimental data to give a reasonable answer to this question. Another question is: how complex must a network machine of a certain kind be to solve a given problem, e.g. how to define the complexity of a problem for a given network type, similar as for Turing Machines or for other types of computers. A great advantage of the Boltzmann Machine as of the Error Propagation Machine is the simple applicability of parallel working special hardware /4,5/.

9. Acknowledgement

The author want to express his gratitude to the initiator of these investigations, Prof. W. Ebeling, for the steady support and stimulation of this work. Thanks also to Dr. T. Boseniuk for many fruitful discussions.

10. References

/8/ Ebeling, W.: Selbsterorganisation und Information Processing by Networks (contributions to this volume)