知識工学と人工知能 66 － 1
（１９８９．９．２２）

ニューラルネットワークのスケジューリング問題への応用

エド ビアジオニ　　阿部 哲也　　石井 暁
（株）東芝　情報通信システム技術研究所

配車計画作成問題を３種類のニューラルネットワークアルゴリズムで解いたので報告する。得られた結果について、ヒューリスティックを用いた既存のエキスパートシステムの出す結果と比べた。バックプロパゲーションを用いた場合、速度は向上したが評価値は少し悪くなった。ホップフィールドネットワークを用いた場合、速度も遅く、評価値も悪かった。シミュレーテッドアニーリングを用いた場合、速度は遅かったが、評価値は最も良い結果が出た。これらの結果を基にして、エキスパートシステムにおけるニューラルネットワークの利用について検討を加える。

APPLYING NEURAL NETWORKS TO SCHEDULING PROBLEMS

Edoardo Biagioni, Tetsuya Abe, Satoru Ishii
Information and Communication Systems Laboratory, TOSHIBA
Yanagicho 70, Saiwai ku, Kawasaki 210, Japan

We have used three different Neural Network Algorithms to re-implemented an existing expert system which schedules trucks to carry loads, where each load's start and end times are known in advance. We compare the performance of the Neural Networks to that of the expert system: the program based on Back Propagation performs acceptably, the program that used a Hopfield Network is unreliable and yields marginal results, and the program that solved the problem using Simulated Annealing performs quite satisfactorily. We generalize our results and analyze useful ways of integrating Neural Networks into expert systems or replacing expert systems by programs based on Neural Networks.

1 Introduction

The problem of scheduling resources subject to a particular constraint occurs frequently and is of considerable economic significance. Most such problems are NP-complete, which means that it is not feasible to find optimal solutions for problems of realistic size. A typical scheduling problem might be finding an assignment of available drivers to trucks such that all scheduled loads reach their destination on time. Another scheduling problem might be assigning airplanes to routes such that all scheduled routes are covered, while minimizing costs.

The usual solution to these problems is approximate: various heuristic procedures are used to find a solution that is good enough for actual use, even though it may be less than optimal.

Over the last few years there have been substantial new developments in the field of Neural Networks. The most interesting recent development is perhaps that of the Back Propagation algorithm [Rumelhart]; this algorithm can be used to recognize and discriminate among complex patterns. Other Neural Network algorithms, such as Hopfield Networks [Hopfield], provide effective solutions to optimization problems. All this has renewed interest in Neural Networks; many interesting problems are still being explored, but Neural Networks are already used almost routinely in many fields, especially pattern recognition (see for example [Dietz] or [Fukushima]).

One direction that has received only slight attention [Gallant] has been the use of Neural Networks in implementing expert systems. Also, the use of Neural Network algorithms in solving optimization problems still seems to be mostly theoretical. Furthermore there has been, to the best of our knowledge, no attempt to solve scheduling problems using Neural Networks. The research reported here addresses all of the above points.

2 Scheduling Problems

A scheduling problem can be defined as a problem in which resources have to be allocated to satisfy given time constraints in such a way that a given cost function can be minimized. The cost function usually reflects the actual direct and indirect costs of the solution, so that the minimum-cost solution is by definition the most desirable solution. For example, using a telephone line from 3 a.m. to 4 a.m. may be cheaper than from 10 a.m. to 11 a.m., but may also be less productive.

In our research we have attempted to solve a particular scheduling problem using various Neural Network algorithms. The problem is a real-world problem with all the accompanying complexities. The real problem was the one actually solved, but in this paper we will only describe a simplified, analogous problem.

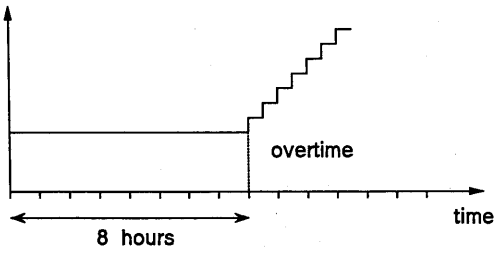

Figure 1. Cost of one truck plotted agains time.

2.1 Truck Scheduling

Our (simplified) test problem is to allocate trucks to carry given loads at given times. The cost of a truck includes a fixed charge that covers the first 8 hours

of use; after that the cost of the truck increases after every additional 1/2 hour because the driver earns overtime pay (see Figure 1). In this particular problem, the trucks are rented from a separate company so the cost of using a truck is known exactly and the number of trucks used does not affect the cost of an individual truck. Each load has scheduled starting and end times which are sufficiently reliable that the schedule can be based on them.

An optimal solution is achieved by keeping each truck busy for most of its basic 8 hours, and do as little overtime as possible while keeping the number of trucks as low as possible. A truck can only carry one load at a time; the problem is how many trucks to schedule, and which loads to assign to which trucks.

2.2 Truck Expert System

A straightforward way to solve this problem is to consider the loads one at a time, beginning with the ones that have the earliest scheduled start times, and assigning them either to a truck that has already been scheduled or to a new one, depending on which of the two solutions is cheaper. This method is fast and simple, but the results are not really good, since the solution is usually rather expensive. Notice that a solutions that is only 3% more expensive than necessary, for example, can be so expensive as to be useless in real life.

A slower method that produces better results is to run the above algorithm as a first step, then try and transfer loads from one truck to another; after several iterations, a state is reached whereby any further change would produce a more expensive solution. This is analogous to 2-opting, a heuristic for solving the traveling salesman problem which exchanges the positions of two cities on the tour whenever

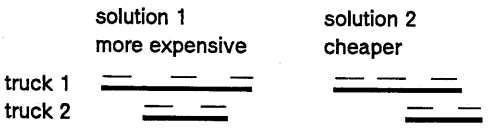

Figure 2. Two different schedules for the same problem: the schedule on the left is more expensive because the first truck is needed for more than 8 hours.

that produces a cheaper tour. Figure 2 shows how one of the heuristics can produce a cheaper solution by swapping two of the jobs.

The above method was implemented and indeed produced satisfactory results in a reasonable amount of time. This program was used as a benchmark for evaluating the implementations of all the Neural Network systems. We shall refer to it as the Expert System, since it uses heuristics that resemble the rules used by human experts performing the same task.

3 Hopfield Network

Since a scheduling problem is just one kind of optimization problem, we tried to solve our truck scheduling problem using a Hopfield Network. A Hopfield Network as described in [Hopfield] is a collection of neurons modeled as nonlinear amplifiers. The input of each neuron is the sum of the outputs of all other neurons, each multiplied by a different weight. At the beginning the amplifiers are in a neutral state and the system evolves until it reaches a stable state in which all of the amplifiers are saturated. The weights must be such that the system always converges to a final stable state where all amplifiers are saturated.

3.1 Problem Encoding

The hardest part of using a Hopfield

Network is finding a good way to encode the problem using a network of neurons. The encoding must be such that when all neurons are saturated and the network is stable, the resulting bits can be decoded to give a solution to the problem.

A typical problem that can be solved with Hopfield Networks is that of the traveling salesman. A traveling salesman problem involving n cities can be solved by a matrix of $n \times n$ neurons. Each column of neurons corresponds to one city and encodes the city's position in the tour, by allowing just one of the neurons to fire (be high). Since no two cities may occupy the same position on the tour, once the final stable state is reached only one neuron of each row is allowed to be high.

It is only the final state that is constrained this way; in the search for a solution, it is perfectly normal for more than one neuron of a row or of a column to fire at the same time. What insures that the constraints are not violated in the final solution are the weights on the connections between neurons: each neuron strongly inhibits all other neurons in the same row or column, so that any state such that a row or column has more than one active neuron is unstable.

3.2 Hopfield Network Truck Scheduling

We used a conceptually similar system to solve our truck scheduling problem. Each column of neurons represents one truck load. Each row of neurons represents one truck, so that a neuron firing implies that the truck represented by the row is assigned to carry the load represented by the column to which the neuron belongs. As in the traveling salesman problem, the solution must be such that exactly one neuron fires in each column, meaning that exactly one truck is scheduled to carry each load.

What is not the same as in the traveling salesman problem is that more than one neuron may fire for each row, since each truck may carry more than one load. This is reflected in the weights between neurons: each neuron always inhibits neurons in the same column, but inhibits only some of the neurons in the same row, and excites most of the others.

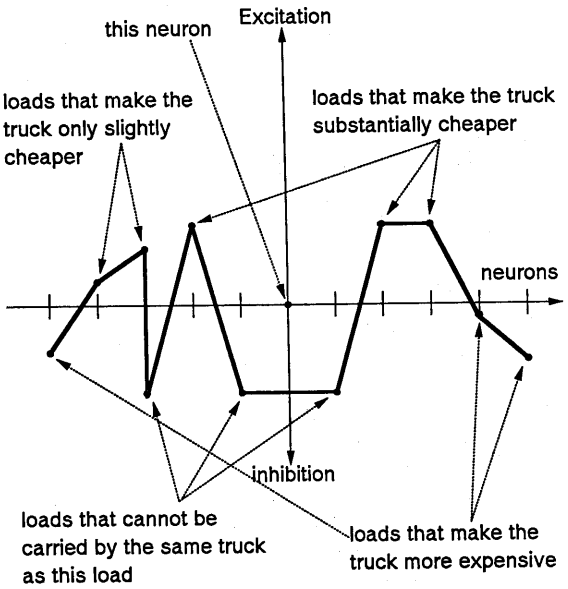

Figure 3. Strength of excitatory and inhibitory connections between neurons in one row. Neurons further to the right represent loads that begin later in the day.

The weights connecting one neuron to other neurons in the same row can be seen in Figure 3. In this graph the neurons are arranged so that loads that are scheduled to start earlier in the day are to the left of the loads that begin later in the day. The neuron for which this graph is drawn corresponds to the vertical axis. Some of the loads with starting time near that of this

neuron overlap it in time, so they are inhibited from firing in the same row (being carried in the same truck) as this neuron.

For all other loads, the closer they are in time the more desirable it is that they be in the same truck; those that are too far apart in time are more likely to incur high overtime cost, so they inhibit each other to some extent. Inhibition or excitation is mutual and symmetric, so whenever one neuron inhibits another, the other one inhibits the first one.

3.3 Optimizations

Hopfield Networks for real problems such as this one tend to be rather large: several of our examples have over 50 loads to schedule, and if we make 30 trucks available, the Neural Network consists of 1,500 neurons. Since each neuron is connected to every other one, there are 2,250,000 connections which on every iteration are multiplied by as many weights.

To speed up the process, we note that the weights for all neurons that are not in the same row or column are constant. We take advantage of this and only actually compute the sum of the weighted inputs from the neurons on the same row and column; at the end we add to this partial total the weighted sum of the outputs of all the neurons in the matrix.

This means that in our example we only have to perform about 80 multiplications and additions per neuron instead of 1,500, giving a speedup of about 20. Without this speedup, the experiment would have been very difficult and time consuming, and practical use would have been unthinkable. Also, the fact that only 1/20 of the weights need to be stored makes it noticeably easier to run the program on a workstation.

4 Simulated Annealing

Another algorithm that can be applied to optimization problems is Simulated Annealing [Kirkpatrick]. Simulated Annealing is not strictly a Neural Network algorithm, since it does not model a network of neurons, but it resembles many other Neural Network algorithms and is one component of some Neural Network algorithms [Hinton].

It shares many mathematical properties with Neural Network algorithms: Neural Networks are a mechanism for minimizing particular energy functions in spaces with a high number of dimensions, and the same can be said for Simulated Annealing. Simulated Annealing is based on the same principles and basic algorithms and behaves very much like many Neural Network Algorithms.

Simulated Annealing finds very low minima of a given energy function by repeatedly making a random change in the system and looking at the cost of the result. On each cycle the change is accepted if the cost has decreased; if the cost has increased, the change is accepted with a probability that depends on the current temperature of the system. If the temperature is high, the change is likely to be accepted. At an intermediate temperature, only changes that increase the cost by a small amount are likely to be accepted. At low temperatures, any change that increases the cost is unlikely to be accepted.

When the temperature is very low, the behavior of Simulated Annealing is similar to that of 2-opting and of the Expert System for the truck scheduling problem. The temperature starts out high (a *molten* state) and is slowly decreased until no more changes are accepted (a *frozen* state).

4.1 Annealing the Truck System

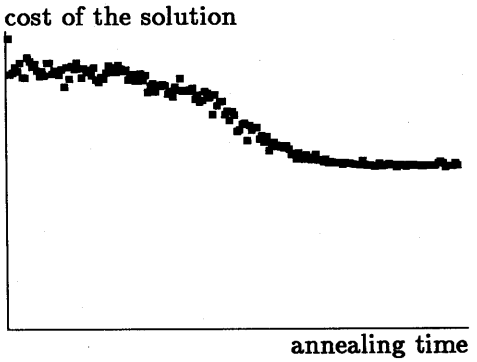

Figure 4. Cost of the solution found by Simulated Annealing plotted against time. The interval in which the slope is greatest is the time at which the system is said to be *freezing*.

Simulated Annealing of the truck system is very simple. The basic step is to move a load, picked at random, from the truck that currently carries it to a different truck, chosen at random. Illegal moves are not attempted, but other than that any move can be tried. The temperature was decreased exponentially, and the cost of the system dropped very noticeably between two specific temperatures (see Figure 4), showing that the system was indeed *freezing* at that point [Kirkpatrick].

4.2 Performance

The results of the Simulated Annealing system were very satisfactory. This system was the only one that consistently produced better results than the Expert System, though each of the other systems did so on occasion. Simulated Annealing is unfortunately rather slow, since the solution is reached by performing random changes from a random starting point; the speed was comparable to that of the optimized Hopfield Network, which on each step had to compute the values of a large number of connections.

5 Back Propagation

In a completely different experiment, we used the pattern-matching capabilities of Back Propagation Networks to improve the Expert System. In the experiments described above, we created a brand new system for each experiment; in this test, the Back Propagation Network was made to work together with the Expert System.

The Expert System, as we have already mentioned, first performs a reasonable, but usually not optimal, assignment of loads to trucks, then uses 4 heuristics to try and find a lower-cost solution. Each of the 4 heuristics returns a list of changes, each of which would reduce the cost: for instance, one heuristic picks two loads and assigns each to the other's truck, and returns all the pairs of loads that can be exchanged to yield a cheaper solution.

Usually the heuristics suggest many different changes; the Expert System picks the one that gives the greatest savings before running the 4 heuristics all over again. The result is usually satisfactory, but running all the heuristics can be slow for large problems, when each one of a large number of changes could give a slight cost improvement.

5.1 Selecting Heuristics

One way to speed up the Expert System is to let only one of the heuristics produce a list of changes. The problem is picking the most profitable heuristic of the 4. Examination of the program running on real data showed that even for a single problem, different heuristics yield the best change at different times. The obvious conclusion is that picking the right heuristic can be difficult.

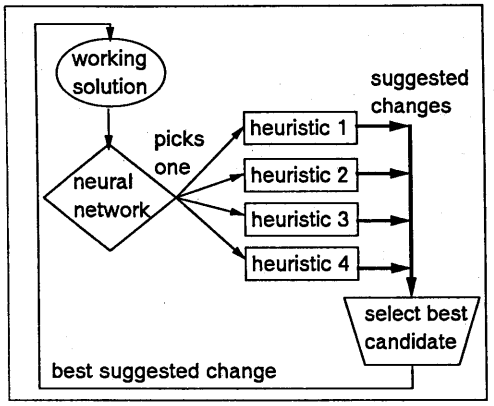

Figure 5. A Back-Propagation Neural Network selects one heuristic within the Expert System; the heuristic produces a list of changes each of which would make the system less expensive.

In the experiment we used a Back Propagation Neural Network to select one heuristic on each optimization step; this heuristic would then be the only one to produce a list of possible changes. Figure 5 shows a block diagram of the heuristic part of the Expert System integrated with the Neural Network. Since it is not clear under what conditions one heuristic will give better results than another, deciding what input to give the Neural Network is perhaps the most difficult problem.

One very direct problem was encoding the truck information for input into the Neural Network: a Back Propagation Neural Network has a fixed number of inputs, whereas the number of loads changes from problem to problem and the number of trucks even changes from iteration to iteration. The only solution that we were able to find was to use averages and cumulative data, e.g. the average number of loads each truck carries, the total cost of the configuration, and so on.

5.2 Effectiveness of Selection

We produced a data set by running the expert system and keeping a record of which heuristic produced the cheapest change at each step. We trained several different networks on this training set and the one that seemed to work best had 51 input nodes, 50 nodes on the first hidden layer, 35 on the second hidden layer, 15 nodes on the third hidden layer, and 4 output nodes. Even though this was the best network tested, actual runs on non-test data showed that the combination of the Neural Network and the Expert System is somewhat less effective than the Expert System alone. The system is indeed faster, taking a little over half the time, but the results are about 2% more expensive than those of the Expert System, which is not satisfactory.

A more effective system can be obtained by having the Neural Network select the heuristic until the chosen heuristic fails to suggest useful changes, then running all 4 heuristics as in the Expert System. This works rather well, since at the beginning, when it matters less which heuristic is used, the Neural Network is allowed to make mistakes, but also to speed up the program; once the solution is close to optimal, all 4 heuristics are applied in trying to make it as good as possible.

This program is only about 10% faster than the Expert System, indicating that the Neural Network is only making a small difference; the final results are usually the same as those of the Expert System, so in this case the improvement in speed is not achieved at the expense of a less optimal result. In other words, using the Neural Network to make the early, non-crucial choices seems useful.

As a control on the usefulness of the Neural Network, we replaced the Neural

Network by a mechanism that would always select the same heuristic. The run time was less, as might be expected, but the results were nearly as good as those obtained using the Neural Network. This leads us to the conclusion that the Back Propagation Neural Network is not particularly useful in this application, though it does make a small difference.

5.3 Back Propagation Expert Systems

As another control, this time to prove that Back Propagation Networks can be useful in writing Expert Systems, we wrote a toy travel advisory Expert System which was trained on a set of actual trips taken and, given distance, travel time, and available modes of transportation, makes suggestions as to what kind of transportation should be used.

Since this system worked satisfactorily, the conclusion is that somehow nature of the truck scheduling problem led to the poor performance of the Neural Network. One obvious problem is that the summary of the truck load information is evidently not sufficient for reliably identifying which heuristic will pick the change which leads to the cheapest solution.

The generalization of this is that it is hard to use a Back Propagation Network to detect patterns in data that does not have a fixed size. If the amount of data is fixed and the data can be directly input into the Neural Network, one would expect the performance of the Back Propagation Network to improve substantially. Another thing to keep in mind is that Back Propagation Networks are effective in making good decisions, but not very effective at making perfect ones — the performance of the Expert System is near-optimal, as far as we can tell, so any system that is compared to this Expert System will look poor unless it is close to perfect.

6 Summary

Altogether we performed three major experiments. Of the three algorithms tested, undoubtedly the most successful was Simulated Annealing. The program based on Simulated Annealing was slower than the Expert System, but consistently produced better results, often by as much as a full percent, which for this problem is a significant difference. This improvement over a system that has been optimized and is judged to perform satisfactorily in actual use is remarkable and indicates that, at least in some cases, some Neural Network algorithms can usefully replace conventional expert systems.

The other experiments were a more qualified success. The Hopfield Network proved to be not only slow but also unreliable, since it was hard to find parameter settings that would consistently produce a valid solution over different problem sizes. Also, the solutions produced by the Hopfield Network were often not as good as those produced by the Expert System. The only case in which a Hopfield Network might be more useful than a Simulated Annealing system is if parallel hardware is available, since Simulated Annealing is harder to implement in parallel than Hopfield Networks are.

The experiment that used Back Propagation yielded a speedup over the Expert System, but the experiment also showed that the pattern matching abilities of the Neural Network were only making a minor difference. However, even in this rather unfavorable case the Network did indeed make a difference, and our experience suggests that if the problem can be encoded in such a way that the patterns become a little clearer, the performance should improve.

6.1 Analysis

These three experiments have shown that different types of algorithms can produce very different results. Simulated Annealing has been very effective in our case, whereas the usefulness of Hopfield Networks and of Back Propagation has been somewhat limited. Our experience shows that the same algorithm applied to different problems may yield rather different results.

One of the reasons that Simulated Annealing worked very well in this case may be the fact that the actual cost function of the system corresponds exactly to the variable being optimized. A further experiment with a different scheduling problem in which it is not possible to compute an exact cost function suggests that for that problem the performance of Simulated Annealing is not as good as the performance seen on the truck scheduling problem.

There also seems to be every reason for believing that more regular scheduling problems might benefit from pattern recognition as provided by Back Propagation Networks.

6.2 Conclusion

As we have seen, there are many practical ways of using Neural Networks instead of conventional expert systems or together with existing systems. The methods described so far are one way of arriving at a solution and are only representative of the many possibilities in this field.

We have also described a few of the problems that may be encountered when attempting to use Neural Network algorithms in the context of expert systems. Altogether, the field seems quite promising and the choice of the right algorithm should make it possible to implement a good scheduling system for many kinds of problems with relatively little effort.

7 References

W. E. Dietz, E. L. Kiech, M. Ali, **Neural Network Models Applied to Real-Time Fault Diagnosis**, *Journal of Neural Network Computing*, Vol. 1, 1 (1989).

Kunihiko Fukushima, **A Neural Network Model for Selective Attention in Visual Pattern Recognition**, *Biological Cybernetics*, Vol 52, 3 (1985), pp. 141–152.

Stephen I. Gallant, **Connectionist Expert Systems**, *Communications of the ACM*, Vol 31, 2 (February 1988), pp. 152–169.

G. E. Hinton, T. J. Sejnowski, **Learning and Relearning in Boltzmann Machines**, in *Parallel Distributed Processing*, by David E. Rumelhart, James L. McClelland, and the PDP research group, *The MIT Press*, 1986, pp. 282–317.

J. J. Hopfield, D. W. Tank, **Neural Computation of Decisions in Optimization Problems**, *Biological Cybernetics*, Vol 52, 3 (1985), pp. 141–152.

S. Kirkpatrick, C. D. Gelatt, Jr., M. P. Vecchi, **Optimization by Simulated Annealing**, *Science*, Vol 220, 4598 (13 May 1983), pp. 671–680.

David E. Rumelhart, Geoffrey E. Hinton, Ronald J. Williams, **Learning representations by back-propagating errors**, *Nature*, Vol 323 (9 October 1986), pp. 533–536.