CS478 – Problem set 4
Solutions

April 24, 2000

1 Problem 4.7 (10pt)

To solve this problem all that has to be done is plug in numbers in algorithm in Table 4.2 in the text with equation T4.5 replaced by Equation 4.18 (to have the version of Back-propagation with moments). Tables 1 and 2 are the the progress of back-propagation for the case when $x_0 = 1$. If you took $x_0 = -1$ the weights after the two steps are depicted in Table 3.

Stage	Var.	Value
Forward	o_c	0.54983
	o_d	0.53866
Backward	δ_d	0.11464
	δ_c	0.00283
Δw	Δw_{ca}	0.00085
	Δw_{cb}	0
	Δw_{c0}	0.00085
	Δw_{dc}	0.01891
	Δw_{d0}	0.03439
new w	w_{ca}	0.10085
	w_{cb}	0.1
	w_{c0}	0.10085
	w_{dc}	0.11891
	w_{d0}	0.13439

Table 1: The values of the state variables of the network for first iteration

2 Problem 4.9

To be able to reproduce the input at the output we need at least 8 different values for the output of the hidden node. Otherwise if two inputs have the same corresponding output in the hidden unit, they have the same output so we

Stage	Var.	Value
Forward	o_c	0.55004
	o_d	0.54978
Backward	δ_d	-0.13608
	δ_c	-0.00400
Δw	Δw_{ca}	0.00076
	Δw_{cb}	-0.00120
	Δw_{c0}	-0.00043
	Δw_{dc}	-0.00543
	Δw_{d0}	-0.00987
new w	w_{ca}	0.10161
	w_{cb}	0.09879
	w_{c0}	0.10041
	w_{dc}	0.11347
	w_{d0}	0.12452

Table 2: The values of the state variables of the network for second iteration

Iter.	Weight	Value
1	w_{ca}	0.10096
	w_{cb}	0.1
	w_{c0}	-0.09904
	w_{dc}	-0.11920
	w_{d0}	0.06159
2	w_{ca}	0.101826
	w_{cb}	0.09888
	w_{c0}	-0.09929
	w_{dc}	0.11775
	w_{d0}	-0.06447

Table 3: The weights after first and second iterations when $x_0 = -1$

don't get the desired behavior. So the activation function for the hidden node should not be the step function (which has only two possible values).

Let the values of the hidden neuron for the 8 different inputs be $v_1, ..., v_8$. Suppose first that the activation function for the output neurons is the step function. The output of the output unit k is 1 if $w_k * o_h + w_{k0} \geq 0$, 0 otherwise, where w_k is the weight of the connection hidden unit-output k, and w_{k0} is the negative of the threshold of output unit k. So the output unit k will have value 1 if $o_h \geq -w_{k0}/w_k$ and 0 otherwise, so it cannot have value 1 if o_h is in some interval, but this is exactly what we need to implement the identity function since say for $o_h = v_3$ we need the 3rd output to be 1 but for the rest of the values to be 0. If v_3 is not the smallest or the biggest value with a step activation function we cannot get this behavior. Since out of 8 possible values 6 cannot have extreme values (they are different as explained before), for 6 of them we will have mistakes, namely the network will say 0 when is supposed to say 1.

The sigmoid function is monotonly increasing. For this reason composing the sigmoid with the step function with threshold 0 gives us exactly the behavior that we can get just with the step function. This means that we cannot approximate the function we are trying to learn on all outputs better than 0.5 on all outputs since otherwise this would mean that we can learn the function just with step activation functions, which is not the case as we showed before. So we make at least a 0.5 error no matter how we pick the weighs so back-propagation has no chance to learn this function.

3 Problem 3

There are two ways to encode the problem as an individual and of course 2 ways to specify the rest of the genetic algorithm (plus variants).

3.1 Per time step task allocation

One way to encode the problem is to talk about what happens in every time step (what task runs on what processor at every time step). 0 can be used as a marker for idle processor, the total number of time steps we have to look at is upper bounded by $\sum_{i=1}^{N} l(t_i)$ (the time it takes to run the processes sequentially). Note that if we have a partial order on the tasks we can always collapse it in a total order so the problem always has at least a legal schedule, so we can talk about the best solution in all the circumstances.

We can take the fitness function to be the last nonzero time slot if the constraints are satisfied (every task runs on a single processor and it takes $l(t_i)$ sequential time slots, the partial order condition holds) and ∞ if one of the constraints doesn't. Note the important property that the fitness function has, namely that the individual where we have a global minimum is the best solution for the original problem (the shortest schedule).

Is not essential to take the fitness function to be ∞ when constraints are not satisfied but picking the initial situation, mutation and crossover is a much

harder task since invalid individuals should be avoided since the fitness function is not total in this case (computes a value only for valid individuals).

If you took the total version of the fitness function, you could have picked almost anything as mutation and crossover, producing invalid individuals not being a concern. Probably such a solution would behave quite poorly in practice.

3.2 Task to process allocation with total ordering

The individuals can also be encoded as m lists of tasks, one for each processor, that denote the order of execution of the tasks on the m processors. The amount of information encoded is smaller but we can make up the rest by using the partial order. The idea is that an encoding corresponds to the best solution that has a supplementary set of constraints (the total ordering per processor and the processor allocation).

In this case the fitness function is much harder to compute since for every encoding you have to solve a minimization problem to compute the time to execute the schedule. The algorithm can scan simultaneously the m lists and simulate the running of the tasks, checking at every step the partial order constraints and running as many things in parallel as possible. Checking each of the constraints is equivalent with a BFS search on the directed graph of partial orders (so O(N)). In the same time the possible invalid encodings (loops in the directed graph) can be detected. As before choosing ∞ as the fitness function for these individuals helps a lot in simplifying the mutation and crossover (which can be almost anything in this case).

Since the hard part is how you compute the fitness function in this solution, I took off 3-4pt for not giving an algorithm for computing it.

3.3 Task to process allocation

One can go even further and just mention in the encoding what task should be run on what processor. To map one such encoding (and to compute the fitness function) one has to find the best solution that has such a task allocation. An algorithm to do this involves two steps: project the partial ordering on the m partitions of tasks, getting m partial orders, and collapse the partial order in a total order (the way you do this is not important); use the algorithm mentioned before to go from total orders per processor to the best solution. The algorithm is polynomial (if care is taking in working with partial orders without computing the closure, using the graph representation for example).

This encoding has the advantage that it doesn't have invalid individuals (if care is taken not to have a task in two lists), any allocation has at least a corresponding solution in the original problem. The disadvantage is the fact that computing the fitness function for an individual is quite time consuming.

The mutation and crossover are very elegant and intuitive for this encoding: mutation=take a random task and move it on a random processor, crossover=combine in any way the allocation lists of the parents and resolve duplicates in any way.

4 Problem 9.4

The $3 \times 2 \times 1$ neural network has 11 weights. Any binary encoding (IEEE encoding for example) can be used to encode the weights and the concatenation of the encodings gives an individual.

Almost any crossover operation is acceptable if you gave a reason for picking this particular one. Just picking one without justification is not good enough.

The main advantage of GA with respect to BP is the fact that the space of hypotheses is better explored, so the chance to get close to the global minimum of the energy function is bigger.

The disadvantage is mainly the speed and the memory requirements.

Some of you mentioned advantages or disadvantages of GA but not with respect to BP and you got 1-4pt off.