Intelligent Applications

التطبيقات الذكية

1. Introduction to Expert Systems

Expert systems are computer programs that are constructed to do the kinds of activities that human experts can do such as design, compose, plan, diagnose, interpret, summarize, audit, give advice. The work such a system is concerned with is typically a task from the worlds of business or engineering/science or government.

Expert system programs are usually set up to operate in a manner that will be perceived as intelligent: that is, as if there were a human expert on the other side of the video terminal.

A characteristic body of programming techniques give these programs their power. Expert systems generally use automated reasoning and the so-called weak methods, such as search or heuristics, to do their work. These techniques are quite distinct from the well-articulated algorithms and crisp mathematical procedures more traditional programming.

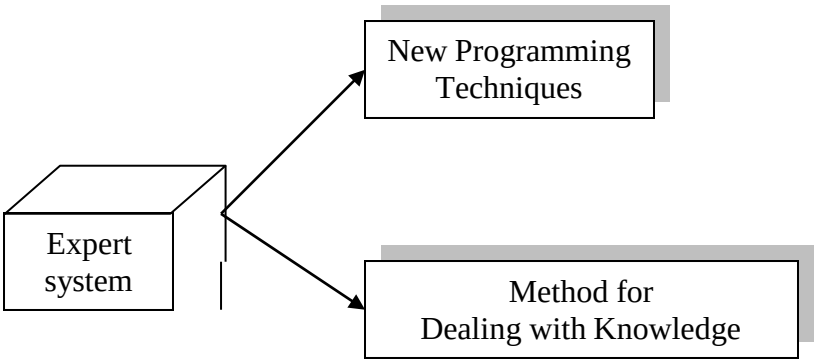


Figure (1) the vectors of expert system development

As shown in Figure(1), the development of expert systems is based on two distinct, yet complementary, vectors:

a. New programming technologies that allow us to deal with knowledge and inference with ease.
b. New design and development methodologies that allow us to effectively use these technologies to deal with complex problems.

The successful development of expert systems relies on a well-balanced approach to these two vectors.

2. Expert System Using

Here is a short nonexhaustive list of some of the things expert systems have been used for:

- To approve loan applications, evaluate insurance risks, and evaluate investment opportunities for the financial community.
- To help chemists find the proper sequence of reactions to create new molecules.
- To configure the hardware and software in a computer to match the unique arrangements specified by individual customers.
- To diagnose and locate faults in a telephone network from tests and trouble reports.
- To identify and correct malfunctions in locomotives.

- To help geologists interpret the data from instrumentation at the drill tip during oil well drilling.
- To help physicians diagnose and treat related groups of diseases, such as infections of the blood or the different kinds of cancers.
- To help navies interpret hydrophone data from arrays of microphones on the ocean floor that are used t\u the surveillance of ships in the vicinity.
- To figure out a chemical compound's molecular structure from experiments with mass spectral data and nuclear magnetic resonance.
- To examine and summarize volumes of rapidly changing data that are generated too last for human scrutiny, such as telemetry data from landsat satellites.

Most of these applications could have been done in more traditional ways as well as through an expert system, but in all these cases there were advantages to casting them in the expert system mold.

In some cases, this strategy made the program more human oriented. In others, it allowed the program to make better judgments.

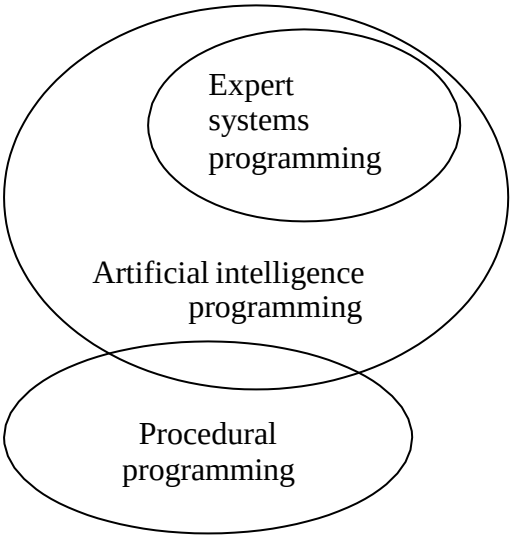
In others, using an expert system made the program easier to maintain and upgrade.

3. Expert Systems are Kind of AI ftrograms

Expert systems occupy a narrow but very important corner of the entire programming establishment. As part of saying what they are, we

need to describe their place within the surrounding framework of established programming systems.

Figure(2) shows the three programming styles that will most concern us. Expert systems are part of a larger unit we might call AI (artificial intelligence) programming. Procedural programming is what everyone learns when they first begin to use BASIC or PASCAL or FORTRAN. Procedural programming and A.I programming are quite different in what they try to do and how they try to do it.



Figure(2) three kinds of programming

In traditional programming (procedural programming), the computer has to be told in great detail exactly what to do and how to do it. This style has been very successful for problems that are well defined. They usually are found in data processing or in engineering or scientific work.

AI programming sometimes seems to have been defined by default, as anything that goes beyond what is easy to do in traditional procedural

programs, but there are common elements in most AI programs. What characterizes these kinds of programs is that they deal with complex problems that are often poorly understood, for which there is no crisp algorithmic solution, and that can benefit from some sort of symbolic reasoning.

There are substantial differences in the internal mechanisms of the computer languages used for these two sorts of problems. Procedural programming focuses on the use of the assignment statement (" = " or ":-") for moving data to and from fixed, prearranged, named locations in memory. These named locations are the program variables. It also depends on a characteristic group of control constructs that tell the computer what to do. Control gets done by using

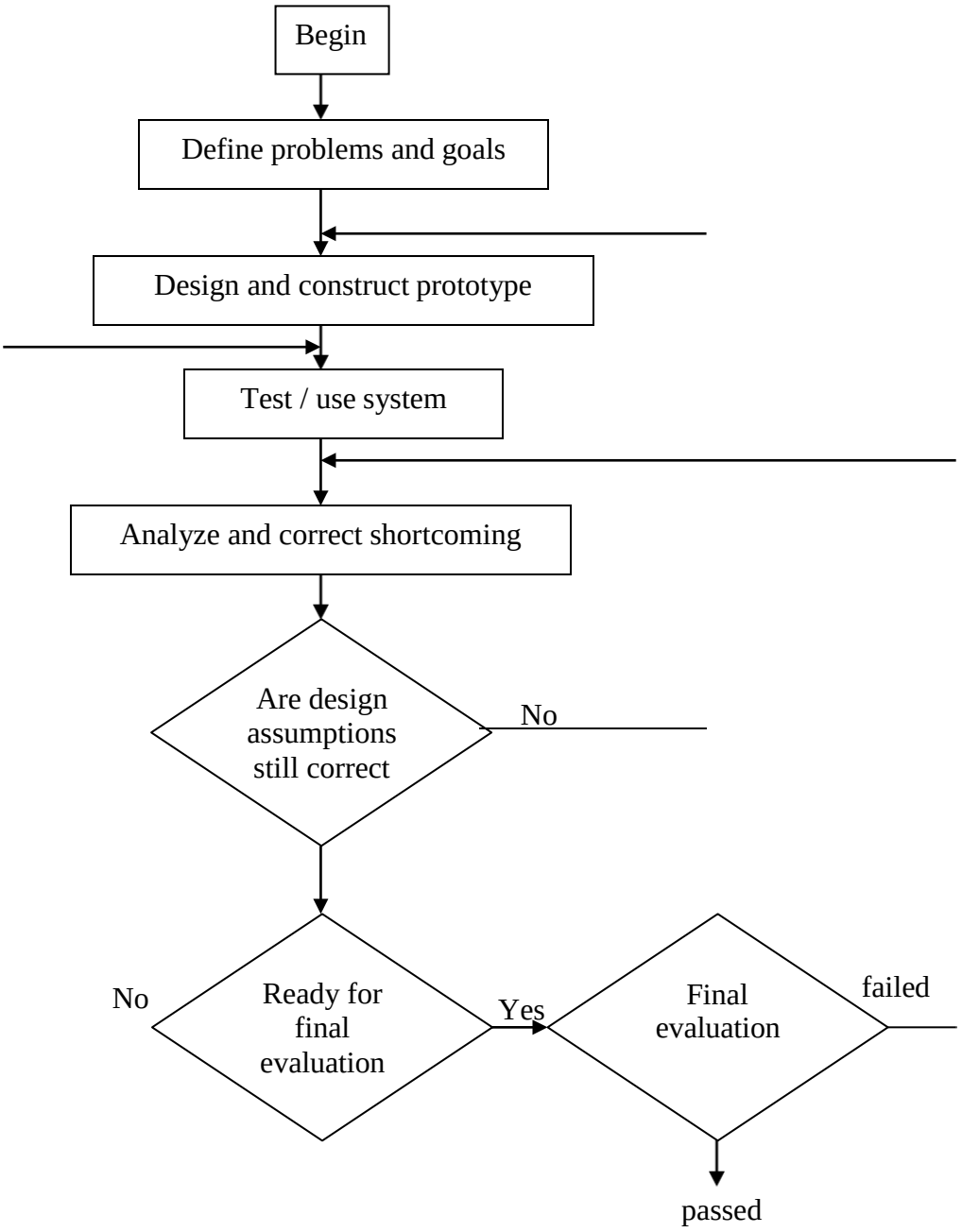
if-then-else	goto
do-while	procedure calls
repeat-until	sequential execution (as default)

AI programs are usually written in languages like Lisp and Prolog. Program variables in these languages have an ephemeral existence on the stack of the underlying computer rather than in fixed memory locations. Data manipulation is done through pattern matching and list building. The list techniques are deceptively simple, but almost any data structure can be built upon this foundation. Many examples of list building will be seen later when we begin to use Prolog. AI programs also use a different set of control constructs. They are :

procedure calls
sequential execution
recursion

4. Expert System, Development Cycle

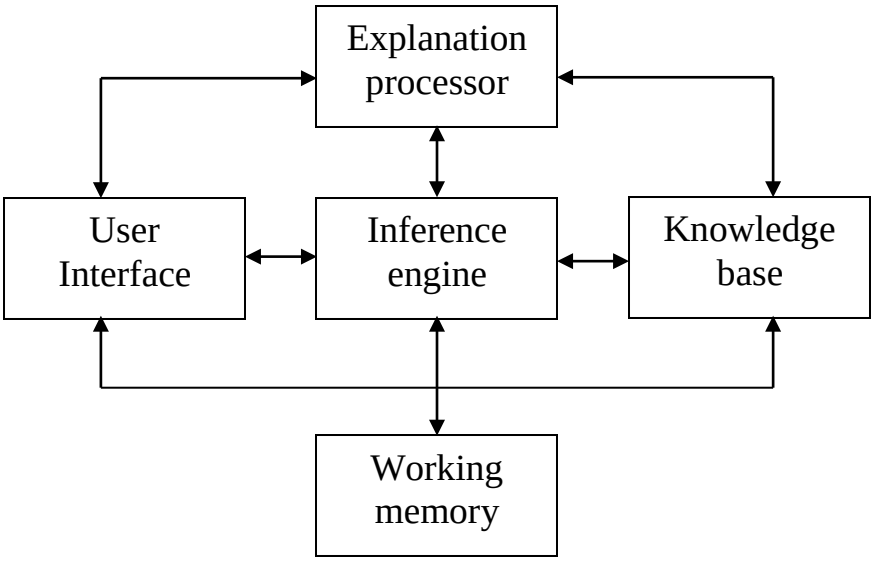
The explanation mechanism allows the program to explain its reasoning to the user, these explanations include justification for the system's conclusions, explanation of why the system needs a particular piece of data. Why questions and How questions. Figure (3) below shows the exploratory cycle for rule based expert system.



Figure(3) The exploratory cycle for expert system

5. Expert System Architecture and Components

The architecture of the expert system consists of several components as shown in figure (4) below:



Figure(4)Expert system architecture

5.1. User Interface

The user interacts with the expert system through a user interface that make access more comfortable for the human and hides much of the system complexity. The interface styles includes questions and answers, menu-driver, natural languages, or graphics interfaces.

5.2. Explanation ftrocessor

The explanation part allows the program to explain its reasoning to the user. These explanations include justifications for the system's conclusion (HOW queries), explanation of why the system needs a particular piece of data (WHY queries).

5.3. Knowledge Base

The heart of the expert system contains the problem solving knowledge (which defined as an original collection of processed information) of the particular applications, this knowledge is represented in several ways such as if-then rules form.

5..4 Inference Engine

The inference engine applies the knowledge to the solution of actual problems. It s the interpreter for the knowledge base. The inference engine performs the recognize act control cycle.

The inference engine consists of the following components:-

1. Rule interpreter.
2. Scheduler
3. HOW process
4. WHY process
5. knowledge base interface.

5.5. Working Memory

It is a part of memory used for matching rules and calculation. When the work is finished this memory will be raised.

6. Systems that Explain their Actions

An interface system that can explain its behavior on demand will seem much more believable and intelligent to its users. In general, there are two things a user might want to know about what the system is doing. When the system asks for a piece of evidence, the user might want to ask,

"Why do you want it?"

When the system states a conclusion, the user will frequently want to ask,

"How did you arrive at that conclusion?"

This section explores simple mechanisms that accommodate both kinds of questioning. HOW and WHY questions are different in several rather obvious ways that affect how they can be handled in an automatic reasoning program. There are certain natural places where these questions are asked, and they are at opposite ends of the inference tree. It is appropriate to let the user ask a WHY question when the system is working with implications at the bottom of the tree; that is: when it will be necessary to ask the user to supply data.

The system never needs to ask for additional information when it is working in the upper parts of the tree. These nodes represent conclusions that the system has figured out. rather than asked for. so a WHY question is not pertinent.

To be able to make the conclusions at the top of the tree, however, is the purpose for which all the reasoning is being done. The system is trying to deduce information about these conclusions. It is appropriate to ask a HOW question when the system reports the results of its reasoning about such nodes.

There is also a difference in timing of the questions. WHY questions will be asked early on and then at unpredictable points all throughout the reasoning. The system asks for information when it discovers that it needs it. The. time for the HOW questions usually comes at the end when all the reasoning is complete and the system is reporting its results.

```
domains

rxnlist = reactions*.

reactions = rxn(symbol, ls, integer, integer).

ls = symbol*.

chemicalList= chemicalForm*.

chemicalForm= chemical(symbol, rxnList, integer, integer).

Li= integer*.

predicates

rxn(symbol, ls, integer, integer).

rawmaterial(symbol, integer, integer).

chemical(symbol, rxnlist, integer, integer).

all_chemical(symbol, chemicalList).

best_chemical(symbol, chemicalForm).

one_chemical(symbol, chemicalForm).

append(rxnlist, rxnlist, rxnlist).

min(chemicalList, chemicalForm).

run(symbol).

clauses

rxn(a, [b1, c1], 12, 60).

rxn(b1, [d1, e1], 5, 45).

rxn(c1, [f1, g1], 3, 15).


rxn(a, [b2, c2], 10, 50).

rxn(b2, [d2, e2], 2, 20).

rxn(c2, [f2, g2], 6, 30).
```

```
rawmaterial(d1, 2, 0).

rawmaterial(e1, 0, 0).

rawmaterial(f1, 2, 0).

rawmaterial(g1, 0, 0).


rawmaterial(d2, 0, 0).

rawmaterial(e2, 1, 0).

rawmaterial(f2, 1, 0).

rawmaterial(g2, 0, 0).


chemical(Y, [], Cost, Time):- rawmaterial(Y, Cost, Time).

chemical(Y, L, Ct, T):-

                rxn(Y, [X1, X2], C, T1), chemical(X1, L1, C1, T2), chemical(X2, L2, C2, T3),

                append(L1, L2, Q), Ct = C+C1+C2,

                T = T+T2+T3, append([rxn(Y, [X1, X2], C, T1)], Q, L).


best_chemical(Y, M):- all_chemical(Y, X), min(X, M).


all_ chemical(Y, X):- findall(S, one_chemical(Y, S), X).


one_chemical(Y, chemical(Y, L, Ct, T)):- chemical(Y, L, Ct, T).


append([], L, L):-!.

append([H|T], L, [H|T1]):- append(T, L, T1).
```

```
min([chemical(Y, L, Ct, T)], chemical(Y, L, Ct, T)).

min([chemical(Y, L, Ct, Time)|T], chemical(Y, L, Ct, Time)):-

                    min(T, chemical(Y1, L1, C1, Time1)), Ct <= C1.

min([chemical(Y, L, Ct, Time)|T], chemical(Y, L2, Ct2, Time2)):-

                    min(T, chemical(Y, L2, Ct2, Time2)), Ct2 <= Ct.


run(X):- write(" chemical synthesis is:"), nl, chemical(X, L, Cost, Time),

        write(L, "\n with total cost =", Cost, " Time =", Time), nl, fail.

run(X):- write("\n Best chemical synthesis:"), nl, best_chemical(X, Y), write(Y), nl.

Goal: run(a).

chemical synthesis:

[rxn("a", ["b1", "c1"], 12, 60), rxn("b1", ["d1", "e1"], 5, 45), rxn("c1", ["f1", "g1"], 3, 15)]

with total cost = 24 time = 120

[rxn("a", ["b2", "c2"], 10, 50), rxn("b2", ["d2", "e2"], 2, 20), rxn("c2", ["f2", "g2"], 6, 30)]

with total cost = 20 time = 100

best chemical synthesis :

chemical("a", [rxn("a", ["b2", "c2"], 10, 50) rxn("b2", ["d2", "e2"], 2, 20), rxn("c2", ["f2", "g2"],
6, 30)], 20, 100)
```

Controlling the Reasoning Strategy

Classification Program with Backward Chaining (Bird, Beast, Fish) Version1

```
database
db_confirm(symbol, symbol)
db_denied(symbol, symbol)
clauses
guess_animal :- identify(X), write("Your animal is a(n) ",X),!.

identify(giraffe) :-
                    it_is(ungulate),
                    confirm(has, long_neck),
                    confirm(has, long_legs),
                    confirm(has, dark_spots)
identify(zebra) :-
                    it_is(ungulate),
                    confirm(has, black_strips),!.
identify(cheetah) :-
                    it_is(mammal),
                    it-is(carnivorous),
                    confirm(has, tawny_color),
                    confirm(has, black_spots),!.
```

```
identify(tiger) :-
                    it_is(mammal),
                    it-is(carnivorous),
                    confirm(has, tawny_color),
                    confirm(has, black_strips),!.

identify(eagle) :-
                    it_is(bird),
                    confirm(does, fly),
                    it-is(carnivorous),
                    confirm(has, use_as_national_symbol),!.
identify(ostrich) :-
                    it_is(bird),
                    not(confirm(does, fly)),
                    confirm(has, long_neck),
                    confirm(has, long_legs),!.
identify(penguin) :-
                    it_is(bird),
                    not(confirm(does, fly)),
                    confirm(does, swim),
                    confirm(has, black_and_white_color),!.
```

```
identify(blue_whale) :-
                        it_is(mammal),
                        not(it-is(carnivorous)),
                        confirm(does, swim),
                        confirm(has, huge_size),!.
identify(octopus) :-
                        not(it_is(mammal),
                        it_is(carnivorous),
                        confirm(does, swim),
                        confirm(has, tentacles),!.
identify(sardine) :-
                        it_is(fish),
                        confirm(has, small_size),
                        confirm(has, use_in_sandwiches),!.
identify(unknown).          /* Catch-all rule if nothing else works. */


it-is(bird):-
                confirm(has, feathers),
                confirm(does, lay_eggs),!
it-is(fish):-
                confirm(does, swim),
                confirm(has, fins),!.
```

```
 it-is(mammal):-
                confirm(has, hair),!.
it-is(mammal):-
                confirm(does, give_milk),!.
it-is(ungulate):-
                it-is(mammal),
                confirm(has, hooves),
                confirm(does, chew_cud),!.
it-is(carnivorous):-
                confirm(has, pointed_teeth),!.
it-is(carnivorous):-
                confirm(does, eat_meat),!.

confirm(X,Y):- db_confirm(X,Y),!.
confirm(X,Y):- not(denied(X,Y)),!, check(X,Y).

denied(X,Y):- db-denied(X,Y),!.

Check(X,Y):- write(X, " it ", Y, \ "n"), readln(Reply), remember(X, Y, Reply).

remember(X, Y, yes):- asserta(db_confirm(X, Y)).
remember(X, y, no):- assereta(db_denied(X, Y)), fail.
```

Controlling the Reasoning Strategy

Classification Program with Forward Chaining (Bird, Beast, Fish) Version2

```
database

db_confirm(symbol, symbol)

db_denied(symbol, symbol)

clauses

guess_animal :-
                    find_animal, have_found(X),
                    write("Your animal is a(n) ",X),nl,!.


find_animal:- test1(X), test2(X,Y), test3(X,Y,Z), test4(X,Y,Z,_),!.

Find_animal.


test1(m):- it_is(mammal),!.

test1(n).


test2(m,c):- it_is(carnivorous),!.

test2(m,n).

test2(n,w):- confirm(does, swim),!.

test2(n,n).
```

```
test3(m,c,s):- confirm(has, strips),
                asserta(have_found(tiger)),!.
test3(m,c,n):- asserta(have_found(cheetah)),!.
test3(m,n,l):- not(confirm(does, swim)),
                not(confirm(does, fly)),!.
test3(m,n,n):- asserta(have_found(blue_whale)),!.
test3(n,n,f):- confirm(does, fly),
               asserta(have_found(eagle)),!.
test3(n,n,n):- asserta(have_found(ostrich)),!.
test3(n,w,t):- cofirm(has, tentacles),
                asserta(have_found(octopus)),!.
test3(n,w,n).

test4(m,n,l,s):- confirm(has, strips),
                 asserta(have_found(zebra)),!.
test4(m,n,l,n):- asserta(have_found(giraffe)),!.
test4(n,w,n,f):- confirm(has, feathers),
                 asserta(have_found(penguin)),!.
test4(n,w,n,n):- asserta(have_found(sardine)),!.

it-is(bird):- confirm(has, feathers),
              confirm(does, lay_eggs),!.
```

```
it-is(fish):- confirm(does, swim),
                confirm(has, fins),!.
 it-is(mammal):- confirm(has, hair),!.
it-is(mammal):- confirm(does, give_milk),!.
it-is(ungulate):- it-is(mammal),
                        confirm(has, hooves),
                        confirm(does, chew_cud),!.
it-is(carnivorous):- confirm(has, pointed_teeth),!.
it-is(carnivorous):- confirm(does, eat_meat),!.

confirm(X,Y):- db_confirm(X,Y),!.
confirm(X,Y):- not(denied(X,Y)),!, check(X,Y).

denied(X,Y):- db-denied(X,Y),!.

Check(X,Y):- write(X, " it ", Y, \ "n"), readln(Reply), remember(X, Y, Reply).

remember(X, Y, yes):- asserta(db_confirm(X, Y)).
remember(X, y, no):- assereta(db_denied(X, Y)), fail.
```

Conclusions

1. Code written for backward chaining is clearer. All the rules in version 1 of BBF have a nice declarative reading. They correspond nicely to most people's intuitive idea of how things should be described when they are part of some kind of hierarchy. The description is top down.

2. Code written for backward reasoning is also much easier to modify or expand. It is apparent without much thought what would have to be done to add another animal (class) to the structure: just define it. But it is not always clear where to attach another instance to a forward reasoning rule structure. In fact, if a number of additions have to be made, all the rules may have to be redone to accommodate the additions and at the same time to maintain the same testing efficiency as was there before.

3. Code for the backward reasoning system will be easier to develop in the first place because the built-in inference method in prolog is backward chining.

Study Question

1. Show what would be required to add these two animals to both versions of BBF:

    - The camel, an ungulate with a hump.
    - The unicorn, an ungulate with a single horn.

2. By examining the listings of the BBF programs, calculate the average number of questions that will be asked to identify an animal in the forward chaining versions and in the backward chaining version.

3. Find a set of rules that describe what to do when your computer will not started. Organize the appropriate rules into both a backward chaining and forward chaining systems (version 1 & 2).

Programs that Work under Uncertainty factor

Approximation Reasoning and Bipolar States

Logical Implications

- Simple Implication

ct(c) = ct(e) * ct(imp)

- AND Implication

ct(c) = min(ct(e1), ct(e2)) * ct(imp)

- OR Implication

ct(c) = max(ct(e1), ct(e2)) * ct(imp)

Bipolar Calculation Values

- If ct1(c) is +ve and ct2(c) is +ve (+ +) then

  Ct(c) = (ct1(c) + ct2(c)) - (ct1(c) * ct2(c))

- If ct1(c) is -ve and ct2(c) is -ve (- -) then

  Ct(c) = (ct1(c) + ct2(c)) + (ct1(c) * ct2(c))

- If [ct1(c) is +ve and ct2(c) is -ve (+ -)] or

  [ct1(c) is -ve and ct2(c) is +ve (- +)] then

  Ct(c) = (ct1(c) + ct2(c)) / (1-min(abs(ct1(c))), (abs(ct2(c)))

Reversible and non reversible Rules

Reversible

- If ct(c) is -ve and prefaced by not then Ct(c) is +ve
- If ct(c) is +ve and prefaced by not then Ct(c) is -ve

Non reversible

- If ct(c) is -ve and prefaced by not then Ct(c) is +ve
- If ct(c) is +ve and prefaced by not then Ct(c) = 1- (+ve)

Knowledge Base

- hypothesis_node(C).
- terminal_node(e).
- imp(logic op, rule type, conclusion name, left condition sign, left condition name, right condition sign, right condition name, imp value)

Systems that Explain their Actions

The HOW & WHY Facilities

Consider the following Inference Network (fuzzy net)

If e1 and e2 then c1 (imp= 0.8) rev

If not(e3) or c3 then c2 (imp= 0.9) rev

If e4 and e5 then c3 (imp= 0.8) rev

If c1 or c2 then c4 (imp= 0.8) rev

Answering WHY Questions

S: Type w(why) or give the certainty for node e4

U: w

S: Attempting to establish c3 via the implication

e4 and e5 → c3

Type w(why) or give the certainty for node e4

U: w

S: Attempting to establish c2 via the implication

not e3 or c3 → c2

Type w(why) or give the certainty for node e4

U: w

S: Attempting to establish c4 via the implication

c1 or c2 → c4

Type w(why) or give the certainty for node e4

U: 0.85

(Why Stack Description)

Why stack description is explained through the lecture.

Answering HOW Question

S: Type h(how) nodename, or c (to continue).

U: h c4

S: Concluded c4 with certainty of 0.68 from

c1 or c2 → c4

The rule is reversible

Certainty of c1 is 0.6

Certainty of c2 is 0.85

The certainty of the implication is 0.8

Used alone the rule suggests a certainty of 0.68

Type h(how) nodename, or c (to continue).

U: h c1

S: Concluded c1 with certainty of 0.6 from

e1 or e2 → c1

The rule is reversible

Certainty of e1 is 0.75

Certainty of e2 is 0.65

The certainty of the implication is 0.8

Used alone the rule suggests a certainty of 0.6

Type h(how) nodename, or c (to continue).

Consider the following Production rules:

if not(e3) or e4 then c1	(imp = 1.0)	nrev
if not(e1) and not(e2) then c2	(imp = 0.8)	rev
if c1 or e5 then c3	(imp = 0.7)	nrev
if not(e6) then c4	(imp = 0.9)	nrev
if e7 and e8 then c5	(imp = 0.8)	nrev
if not(e9) then c5	(imp = 0.9)	rev
if c2 then c6	(imp = 0.9)	rev
if c3 then c6	(imp = 0.9)	nrev
if c4 and c5 then c6	(imp = 0.85)	nrev

e1= 0.2 e2= -0.2 e3= -0.2 e4= 0.7 e5= -0.5 e6= -0.8 e7= 0.8

e8= 0.8 e9= -0.7

Systems That Depend on Reasoning under Uncertainty

Approximate Reasoning (Structure of the FUZZYNET Program)

```
driver:- hypothesis-node(X), allinfer(X, Ct),
        write("The certainty for ", X, "is", Ct), nl, fail.

allinfer(Node, Ct):- findall(C1, infer(Node, C1), Ctlist), supercombine(Ctlist, Ct).
```

/*A simple implication */

```
infer(Node1, Ct):-
        imp(s, Use, Node1, Sign, Node2, _, _, C1),
        allinfer(Node2, C2),
        find_multiplier(Sign, Mult, dummy, 0), CS = Mult * C2,
        qualifier(Use, CS, Qmult), Ct = CS * C1 * Qmult.
```

/* An implication with an AND in the Premise */

```
infer(Node1, Ct):-
        imp(a, Use, Node1, SignL, Node2, SignR, Node3, C1),
        allinfer(Node2, C2),
        allinfer(Node3, C3),
        find_multiplier(SignL, MultL, SignR, MultR),
        C2S = MultL * C2, C3S = MultR * C3,
        min(C2S, C3S, CX), qualifier(Use, CX, Qmult), Ct = CX * C1 * Qmult.
```

```
/* An implication with an OR in the Premise */
 infer(Node1, Ct):-
                imp(o, Use, Node1, SignL, Node2, SignR, Node3, C1),
                allinfer(Node2, C2),
                allinfer(Node3, C3),
                find_multiplier(SignL, MultL, SignR, MultR),
                C2S = MultL * C2, C3S = MultR * C3,
                max(C2S, C3S, CX), qualifier(Use, CX, Qmult), Ct = CX * C1 * Qmult.
infer(Node1, Ct):-
                terminal_node(Node1), evidence(Node1, Ct),!.
infer(Node1, Ct):-
                terminal_node(Node1),
                write("What is the certainty for node", Node1),
                nl, readreal(Ct), asserta(evidence(Node1, Ct)),!.

/* This is used for simple implication */
find_multiplier(pos, 1, dummy, 0).
find_multiplier(neg, -1, dummy, 0).
/* This is used for AND and OR implications */
find_multiplier(pos, 1, pos, 1).
find_multiplier(pos, 1, neg, -1).
find_multiplier(neg, -1, pos, 1).
find_multiplier(neg, -1, neg, -1).
```

```
supercombine([Ct], Ct):-!.

supercombine([C1, C2], Ct):- combine([C1, C2], Ct), !.

supercombine([C1, C2|T], Ct):- combine([C1, C2], C3), append([C3], T, TL),
                                    supercombine(TL, Ct), !.

combine([-1, 1], 0).

combine([1, -1], 0).

Combine([C1, C2], Ct):- C1 >= 0, C2>= 0, Ct = C1 + C2 - C1 * C2.

Combine([C1, C2], Ct):- C1 < 0, C2< 0, Ct = C1 + C2 + C1 * C2.

combine([C1, C2], Ct):- C1 < 0, C2 >= 0, absvalue(C1, Z1), absvalue(C2, Z2),
                        min(Z1, Z2, Z3), Ct = (C1 + C2) / (1 – Z3).

combine([C1, C2], Ct):- C2 < 0, C1 >= 0, absvalue(C1, Z1), absvalue(C2, Z2),
                        min(Z1, Z2, Z3), Ct = (C1 + C2) / (1 – Z3).


absvalue(X, Y):- X = 0, Y = 0, !.

absvalue(X, Y):- X > 0, Y = X, !.

absvalue(X, Y):- X < 0, Y = -X, !.


qualifier(Use, C, Qmult):- Use = “r”, Qmult = 1, !.

qualifier(Use, C, Qmult):- Use = “n”, C >= 0, Qmult = 1, !.

qualifier(Use, C, Qmult):- Use = “n”, C < 0, Qmult = 0, !.
```

```
/* For and implication, the other in the same manner */
infer(Node1, Ct):-
        imp(a, Use, Node1, SignL, Node2, SignR, Node3, C1),
        assserta(dbimp(a, Use, Node1, SignL, Node2, SignR, Node3, C1)),
        assserta(tdbimp(a, Use, Node1, SignL, Node2, SignR, Node3, C1)),
        allinfer(Node2, C2),
        allinfer(Node3, C3),
        find_multiplier(SignL, MultL, SignR, MultR),
        C2S = MultL * C2, C3S = MultR * C3,
        min(C2S, C3S, CX), qualifier(Use, CX, Qmult), Ct = CX * C1 * Qmult,
        assertz(infer_summary(
        imp(a, Use, Node1, SignL, Node2, SignR, Node3, C1), Ct)),
        retract(dbimp(a, Use, Node1, SignL, Node2, SignR, Node3, C1)),
        retract(tdbimp(a, Use, Node1, SignL, Node2, SignR, Node3, C1)).

/* How Facility Sub Program */
Exsys_driver :- getallans, showresults,!.

    Getallans :- not(prepare_answer).
    Prepare_answer :- answer(X, Y), fail.
    answer(X, Y) :- hypothesis_node(X), allinfer(X, Y), assert(danswer(X, Y)).
```

```
Showresults :- not(displayall).
        displayall :- display_aoe_answer, fail.
        display_aoe_answer :- danswer(X, Y), clearwindow,
                                write("For this hypothesis:"), nl,
                                write(" ", X),nl, write("The certainty is:", Y),nl, nl,
                                not(how_describer(X)).

how_describer(Node) :- repeat, nl,
                           write("Type h(how) nodename, or c(to continue),"),
                           nl, readln(Reply), nl, how_explain(Reply),!.

how_explain(Reply) :- Reply = "c".
how_explain(Reply) :- fronttoken(Reply, _, X1), fronttoken(X1, X, _),
                       infer_summary(imp(_, _, X, _, _, _, _, _), _), clearwindow,!,
                       write("The rule(s) that bear upon this conclusion are: "),
                       nl, nl, infer_summary(imp(A, A1, X, R, S, C, D, E),F),
                       write("Concluded: ", X), nl, gettype(A, Z),
                       write("from an ", Z), nl, write(" premise 1 was: ",S), nl,
                      write(" premise 2 was: ",D), nl,
                       write("The certainty from use of this rule alone was: ",F),
                      nl, nl, fail.
```

```
how_explain(Reply) :- fronttoken(Reply, _, X1), fronttoken(X1, X, _),
                      terminal_node(X), evidence(X, C),
                      write("You told me that: "), nl, write(" ", X), nl,
                      write("has a certainty of: ",C), nl, fail.
```

/* Why Facility Sub Program */

```
infer(Node, Ct) :- terminal_node(Node), evidence(Node, Ct), !.
infer(Node, Ct) :- terminal_node(Node), repeat, nl,
                   write("Type w(why) or give the certainty for node ", Node),
                   nl, readln(Reply), reply_to_input(Node, Reply, Ct), !.

reply_to_input(Node, Reply, Ct) :- not(isname(Reply)), adjuststack,
                                   str_real(Reply, CT), asserta(evidence(Node,Ct)),!.
reply_to_input(_, Reply, _) :- isname(Reply), Reply = "w", nl,
                               dbimp(U, V, R, S, S1, X, Y, Y1),
                               why_describer(U, V, R, S, S1, X, Y, Y1),
                               retract(dbimp(U, V, R, S, S1, X, Y, Y1)),
                               putadjustflag, pauser, !, fail.

why_describer(U, U1, V, R, S, X, Y, Z) :- clearwindow, nl, U <>"s", gettype(U,UU),
        write("I am trying to use an inference rule of the type "),
        nl, write(UU), write(", to support the conclusion: "), nl,
        write("   ", V), nl, write("Premise 1 is: ",S), nl, getmode(R, RR),
```

```
        write(" This premise will be used ", RR), nl, write("Premise 2 is: ",Y),
        nl, getmode(X, XX), nl, write(" This premise will be used ", XX), nl,
        write("The certainty of the implication is: ", Z), nl, !.
why_describer("s", V1, V, R, S, X, Y, Z) :- clearwindow, nl,
        write("I am trying to use an inference rule of the type "), nl,
        write("simple implication, to support the conclusion: "), nl,
        write(" ", V), nl, write("premise 1 is: ", S), nl, getmode(R, RR),
        write("  This premise will be used ", RR), nl
        write("The certainty of the implication is: ", Z), nl, !.

gettype("a", "and implication").
gettype("o", "or implication").
gettype("s", "simple implication").

Getmode("pos", "as you see it.").
Getmode("neg", "prefaced by not.").
```

Natural Language Interfaces
Formal Method

The people respect clever student.

Clever students can own respecting by their good works.

1. Build the Context Free Grammar for the above sentences.
2. Write a complete prolog program that parse the above sentences using the Context Free Grammar in step 1.

1.

S → Np, Vp, Np / Np, Vp, Np, Pp

Np → det, noun / adj, noun / noun / det, adj, noun

Vp → verb / h.verb, verb

Pp → preposition, Np

2.

```
clauses
run:- readln(S), str_to_list(S, L), parse(L).
parse(L):- append(A1, A2, A3, _, L),
           np(A1),
           vp(A2),
           np(A3).

parse(L):- append(A1, A2, A3, A4, L),
           np(A1),
           vp(A2),
           np(A3),
           pp(A4).

np(X):- append(Y1, Y2, _, _, X),
        det(Y1),
        noun(Y2).
np(X):- append(Y1, Y2, _, _, X),
        adj(Y1),
```

```
            noun(Y2).
np(X):- append(Y1, Y2, Y3, _, X),
            det(Y1),
            adj(Y2),
            noun(Y3).
np(X):- noun(X).

vp(Z):- append(Y1, Y2, _, _, Z),
            h.verb(Y1),
            verb(Y2).
vp(Z):- verb(Z).

pp(M):- append(W1, W2, _, _, M),
             preposition(W1),
             np(W2).

/ * set of Facts */
det([“the”]).     det([“their”]).
noun([“people”]).          noun([“student”]).
noun([“respecting”]).     noun([“work”]).
adj([“clever”]).              adj([“good”]).
verb([“respect”]).          verb([“own”]).
h.verb([“can”]).
preposition([“by”]).
```

Analyzing the semantic structure of a Sentence

Introduction to Thematic Analysis (Case Grammar)

- **Object Case** (is the noun group that receives the action of the verb)
- **Agent Case** (is the entity that applies the action to the object)
- **Co Agent Case** (shares in applying the action that the sentence is about) or (pronoun followed by a noun)

EX: **"The Realtor and his assistant inspected a house for their client ."**

- **Beneficiary Case** (concerns the entity on whose behalf the action in the sentence was Performed) the beneficiary noun group is "for their client"
- **Location Case** (concerns noun group that express where the action took place)
- **Time Case** (this noun group expresses when the action took place)

EX: **"at 5 o'clock"**

- **Instrument Case** (noun group that identifies something used by the agent to apply the action carried by the verb)

EX: **"with the sharp Knife"**

- **Source and Destination Case** (the action sentence frequently is about movement from one place or state to another, these beginning and ending places for the action are associated with source and destination noun groups)

EX: **"The dog chased the insurance agent out of the yard and into his car"**

The source case noun group is **"out of the yard"**

The destination group is **"into his car"**

- **Trajectory Case** (there will be noun groups whose function in the sentence is to describe the path over which the action occurred)

EX: **"The man drove in his car through the woods to his next client"**

- **Conveyance Case** (if the action occurs in some kind of

Carrier, this is a conveyance noun group)

EX: **"in his car"**

Automatic Translation

An Example of the Use of Thematic Analysis (From English Language)

EX: **"Jane repaired the radio for Dan with the test instrument"**

Verb: (to repair)

Verb tense: past tense

Verb Aspect: 3rd person singular (repaired)

Object: the radio

Agent: Jane

Instrument: the test instrument

Beneficiary: Dan

To Germany Language

Verb: (reparieren)

Verb tense: past tense

Verb Aspect: 3rd person singular (hat repariert)

Object: das radio

Agent: Jana

Instrument: die Probeinstrumenten

Beneficiary: Dan

<agent> <verb_first_part> fur <beneficiary> <object> mit <instrument> <verb_second_part>

<agent> <verb_first_part> fur <beneficiary> <object>

Jana hat fur Dan das Radio

mit <instrument> <verb_second_part> mit die Probeinstrumenten repariert

Parts of the Program (Thematic Analysis)

```
sentence(S,S0) :- agent(S,S1), backparta(S1,S0).

backparta(S,S0) :- verb(S,S1), object(S1, S0).

sentence(S,S0) :- agent(S,S1), backpartb(S1,S0).

backpartb(S,S0) :- verb(S,S1), backpartc(S1, S0).

backpartc(S,S0) :- object(S, S1), instrument(S1,S0).

sentence(S,S0) :- agent(S,S1), backpartd(S1,S0).

backpartd(S,S0) :- verb(S,S1), backparte(S1, S0).

backparte(S,S0) :- object(S, S1), backpartf(S1,S0).

backpartf(S,S0) :- trajectory(S,S1), time(S1,S0).
```

Natural Language Interfaces
Informal Method (Dictionary Building)

```
clauses

/* set of facts */

Own(John, B.Sc, 1980, Scientific).
Own(Roy, M.Sc, 1984, technique).
Own(Tomy, B.Sc, 1982, Engineer).
Own(Har, Ph.D, 1978, Scientific).

reject("HOW").
reject("GO").
reject("ALL").
reject("FIND").
reject("THE").
reject("SOME").
reject("I").
reject("HAVE").

dsyn("B.Sc", "B. of Science").
dsyn("M.Sc", "Master of Science").
dsyn("Ph.D", "Philosophy of Doctorate").

docdriver:- repeat, nl, getquery(X), findref(X, Y),
            produceans(Y), fail.

getquery(Z):- write("please ask your question."),
              nl, readln(Y), upper_lower(Y1, Y),
              changeform(Y1, Z).

changeform(S, [H|T]):- fronttoken(S, H, S1), !, changeform(S1, T).
changeform(_, []):-!.

findref(X, Y):- memberof(Y, X), not(reject(Y)), !.
```

```
produceans(X):- own(X, X1, Y, Z), putflag,
                write(X, “has”, X1, “since the year”, Y, “in”, Z),nl.
produceans(X):- syn(X1, W), own(X, W, Y, Z), putflag,
                write(X, “has”, X1, “since the year”, Y, “in”, Z),nl.
produceans(_):- not(flag),
               write(“we have no information on that.”), nl.
produceans(_):- remflag.

putflag:- not(flag), assert(flag),!.
putflag.

remflag:- flag, retract(flag),!.
remflag.

syn(Y,X):- dsyn(X, Y).
syn(Y,X):- dsyn(Y, X).

dsyn(Y,X):- concat(X, “S”, Y).
dsyn(Y,X):- concat(X, “ES”, Y).
dsyn(Y,X):- concat(X, “’S”, Y).
```

Computer Sciences Department

(Software & Computer Security Branches)

AI Applications & Systems - Fourth Class

What heuristics would you use in solving these problems?

1. You are looking for a parking space in a moderately crowded parking lot.
2. You think a particular radio show you want to hear is on now, but you do not know where it is on the dial, and you have no other guidance such as a newspaper listing.
3. You are in a large office building. You are lost, and you want to find the personal office, but you are embarrassed to ask where it is.

Think about an elevator with the following controls: buttons for three floors, buttons to open and close the door, a sensor to see if the door is obstructed, a timer to time how long to leave it open, and single call buttons on each floor. Write a production system that would cause the elevator to operate in the conventional manner if the production system were controlling the operation. Atypical production would be:

If (timer_expired and door_is_open and door_not_obstructed) then (close_door)

Consider the category scheme to classify expert system. For each of the following, discuss if the example would be an expert system at all and, if so, what type:

1. A program to forecast the local weather.
2. A program to reason about what to do when your car will not started.
3. A program for a help-line service where the person answering the phone has to give advice about poisons that someone might have taken.
4. A program to predict what courses to give and how many sections to plan for in the next three semesters in a large college department.

5. A program to determine the best route for a salesperson to take on any given day to visit all his clients and use the minimum amount of gasoline that is possible.
6. A program to produce a 3-dimensional drawing of a house, given a textual description of the arrangement and dimensions of the rooms.

Write a small expert system program to construct optimal restaurant menus that follows the pattern of Student Advisor System.

The chemical synthesis program currently works with reactions like this:

X + y --→ z ……..with cost (c)

1. How would things have to be modified so that reactions like this one could be included in the reaction data base that the program knows about?

   r --→ s …….with cost (c)

   This is anticipating the type of reaction where you treat a chemical in a certain way (heating perhaps) and it turns into something else.
2. How would things have to be modified so that reactions like this one could be included? q + r + s --→ w …….with cost (c)
3. What modification would be necessary for the program to carry along two costs with each synthesis: One might be the reaction cost and the other the length of time the reaction took to complete.
4. What modification would be necessary for the program to include a function that carries the best synthesis among many syntheses?

Computer Sciences Department
(Software & Computer Security Branches)
AI Applications & Systems - Fourth Class

---

1. Try to build again the structure of the fuzzy net program (certainty Program) that accept any arbitrary inference tree.

---

2. Given the following information:

```
index("book1", "OGOFF", ["99"]).
index("book1", "EDLIN", ["41-46", "57"]).
index("book2", "EDLIN", ["100-102"]).
index("book7", "EDLIN", ["100", "110"]).
index("book1", "EXE", ["14-18"]).
index("book1", "ECHO", ["35", "146"]).
index("book7", "BNF", ["51", "55-56"]).
index("book7", "BNF", ["30-31"]).
index("book8", "BNF", ["109", "130-148"]).
index("book4", "RERURSION", ["56-78"]).
index("book7", "RECURSION",["119-125"]).

dsyn("LOGOUT", "LOGOFF").
dsyn("LOGIN", "LOGON").
dsyn("BENEFIT", "ADVANTAGE").
dsyn("PROCESSING", "MANIPULATING").
dsyn("INTELLIGENT", "SMART").
dsyn("FACT", "REAL").

reject("HOW").
reject("ANY").
reject("ABOUT").   and so on
```

Write a complete prolog program to index the above information by using the Dictionary (informal) Natural Language Interface technique.

---

3. Which rules (in the chemical synthesis program) is adjusted when the user asks **how** after the program implementation? Write them.

---

4. Which rules (in the B.B.F program) is adjusted when the user asks **why** when the system asks for any feature? Write them.

University of Technology
Department of Computer Science
Fourth Class (Software & Computer Security Branches)
Lecturer: Dr. Hasanen S. Abdullah
Intelligent Systems & Applications

References

1- Fundamentals of Neural Networks: Architecture, Algorithms and Application. By Laurence Fausett.
2- Genetic Algorithms (Search, Optimization and Machine Learning). By David E. Goldberg.
3- Neural Networks. Fundamentals, Application and Examples. By Werner Kinnebrock.
4- Neural Network for Identification, Prediction and Control. By D. T. Pham and X. Liu.

1.1 Introduction

Artificial neural network (ANN) models have been studied for many years with the hope of achieving "Human-like performance", Different names were given to these models such as:

- Parallel distributed processing models
- Biological computers or Electronic Brains.
- Connectionist models
- Neural morphic system

After that, all these names settled on Artificial Neural Networks (ANN) and after it on neural networks (NN) only.

There are two basic different between computer and neural, these are:

1- These models are composed of many non-linear computational elements operating in parallel and arranged in patterns reminiscent of biological neural networks.
2- Computational Elements (or node s) are connected via weights that are typically adapted during use to improve performance just like human brain.

Computer ⟶ logic Elements (1, 0)
Neural ⟶ weighted performance

1.2 Development of Neural Networks

An early attempt to understand biological computations was stimulated by McCulloch 4 pitts in [1943], who modeled biological neurons as logical as logical decision elements these elements were described by a two – valued state variables (on, off) and organized into logical decision networks that could compute simple Boolean functions.

In 1961 Rosenblatt salved simple pattern recognition problems using perceptrons. Minskey and paert in [1969] studied that capabilities and limitations of perceptrons and concluded that many interesting problems could never be soled by perceptron networks.

Recent work by Hopfield examined the computational power of a model system of two –state neurons operating with organized symmetric connections and feed back connectivity. The inclusion of feed –back connectivity in these networks distinguished them from perceptron – line networks. Moreover, graded – response neurons were used to demonstrate the power * speed of these Networks. Recent interest in neural networks is due to the interest in building parallel computers and most importantly due the discovery of powerful network learning algorithms.

1.3 Areas of Neural Networks

The areas in which neural networks are currently being applied are:

1-signal processing

2- Pattern Recognition.

3-control problems

4-medicine

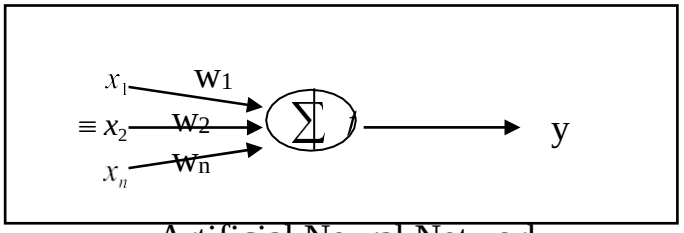
5-speech production

6-speech Recognition

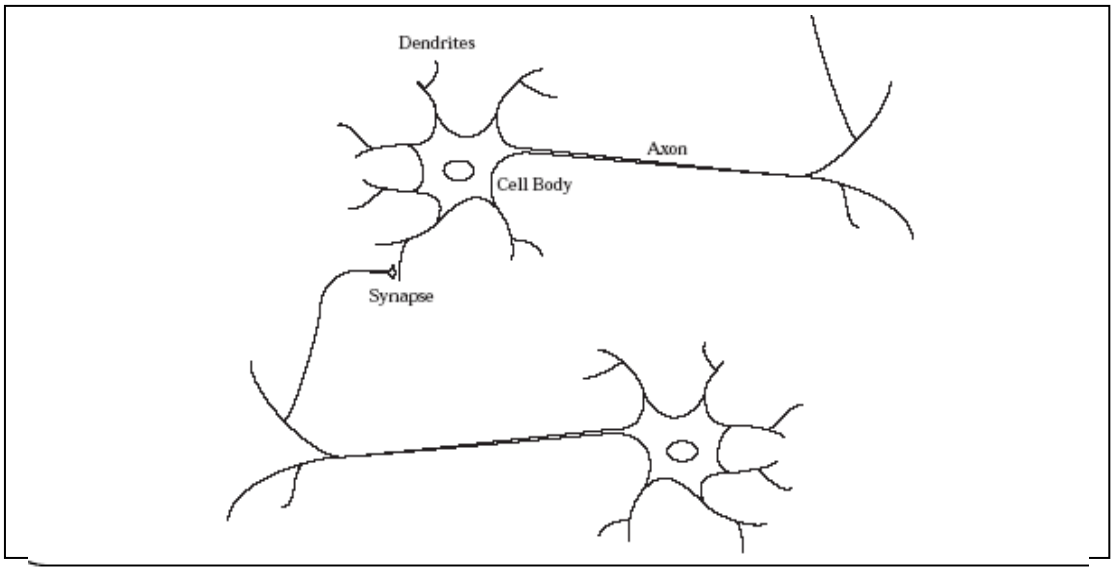
7-Business

2.1 Theory of Neural Networks (NN)

Human brain is the most complicated computing device known to a human being. The capability of thinking, remembering, and problem solving of the brain has inspired many scientists to model its operations. Neural network is an attempt to model the functionality of the brain in a simplified manner. These models attempt to achieve "good" performance via dense interconnections of simple computational elements. The term (ANN) and the connection of its models are typically used to distinguish them from biological network of neurons of living organism which can be represented systematically as shown in figure below



Artificial Neural Network



Biological Neural Network

Neclues is a simple processing unite which receives and combines signals from many other neurons through input paths called ***dendrites*** if the combined signal is strong enough, it activates the firing of neuron which

produces an o/p signal. The path of the o/p signal is called the ***axon***, ***synapse*** is the junction between the (axon) of the neuron and the dendrites of the other neurons. The transmission across this junction is chemical in nature and the amount of signal transferred depends on the synaptic strength of the junction. This synoptic strength is modified when the brain is learning.

Weights (ANN) ≡ synaptic strength (biological Networks)

2.2 Artificial Neural Networks (ANN)

An artificial neural network is an information processing system that has certain performance characters in common with biological neural networks. Artificial neural networks have been developed as generalizations of mathematical models of human cognition or neural biology, based on the assumptions that:-

1-Information processing occurs at many simple elements called neurans.

2-Signals are passed between neurons over connection links.

3-Each connection link has an associated weight which, in a typical neural net, multiplies the signal transmitted.

4-Each neuron applies an action function (usually nonlinear) to its net input (sum of weighted input signals) to determine its output signal.

A Neural network is characterized by:

1- Architecture: - its pattern of connections between the neurons.
2- Training Learning Algorithm: - its method of determining the weights on the connections.
3- Activation function.

2.2.1 Properties of ANN

1-parallelism

2-capacity for adaptation "learning rather programming"

3-capacity of generalization

4-no problem definition

5-abstraction & solving problem with noisy data.

6-Ease of constriction & learning.

7-Distributed memory

8- Fault tolerance

2.3 Learning in Neural Network

In case a neural network is to be used for particle applications, a general procedure is to be taken, which in its various steps can be described as follows:-

1: A logical function to be represented is given. The input vector e_1 , e_2, e_3, …. , e_n are present, whom the output vectors a_1, a_2, a_3, …. , a_n assigned. These functions are to be represented by a network.

2: A topology is to be selected for the network.

3: The weights w_1, w_2, w_3, … are to be selected in such away that the network represents The given function (n) the selected topology. Learn procedures are to be used for determining the weights.

4: After the weights have been learned and the network becomes available, it can be used as after as desired.

The learning of weights is generally done as follows:

1- Set random numbers. For all weights.

2- Select a random input vector ej.

3- Calculate the output vector Oj with the current weights.

4- Compare Oj with the destination vector aj , if Cj = aj then continue with (2).

Else correct the weights according to a suitable correction formula and then continue with (2).

There are ***three type*** of learning in which the weights organize themselves according to the task to be learnt, these types are:-

1. Supervised learning

The supervised is that, at every step the system is informed about the exact output vector. The weights are changed according to a formula (e.g. the delta-rule), if o/p is unequal to a. This method can be compared to learning under a teacher, who knows the contents to be learned and regulates them accordingly in the learning procedure.

2. Unsupervised Learning

Here the correct final vector is not specified, but instead the weights are changed through random numbers. With the help of an evaluation function one can ascertain whether the output calculated with the changed weights is better than the previous one. In this case the changed weights are stored, else forgotten. This type of learning is also called reinforcement learning.

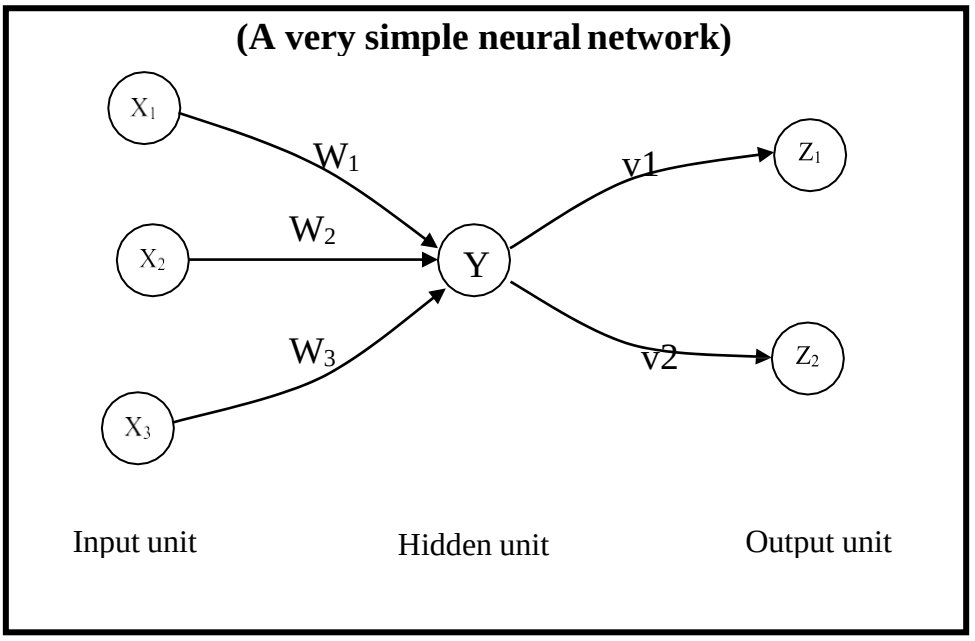
3. Learning through Self- Organization

The weights changed themselves at every learning step. The change depends up on

1- The neighborhood of the input pattern.
2- The probability pattern, with which the permissible input pattern is offered.

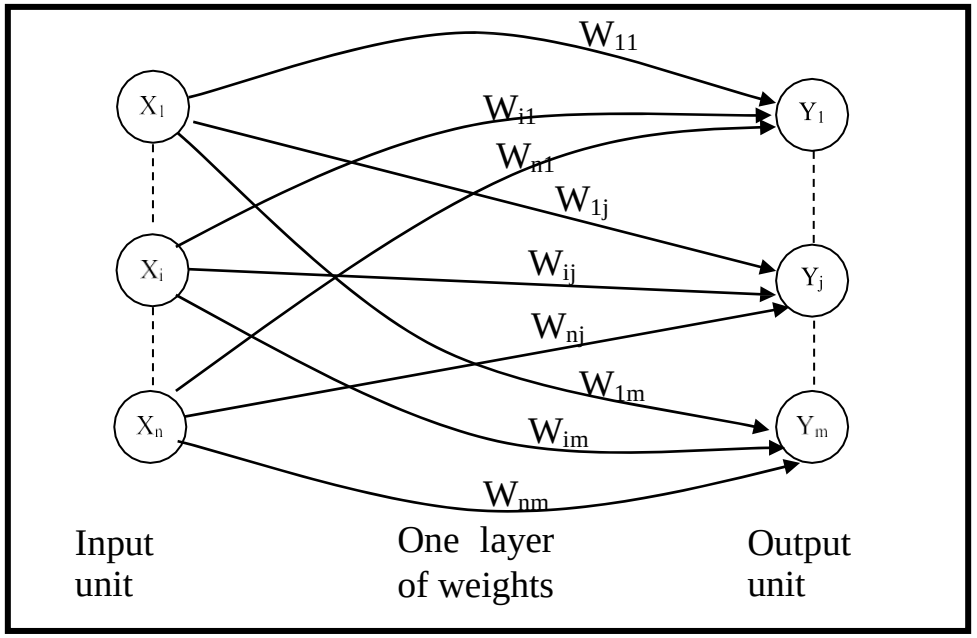
2.4 Typical Architecture of NN

Neural nets are often classified as single layer or multilayer. In determining the number of layers, the input units are not counted as a layer, because they perform no computation. Equivalently, the number of layers in the net can be defined to be the number of layers of weighted interconnects links between the slabs of neurons. This view is motivated by the fact that the weights in a net contain extremely important information. The net shown bellow has two layers of weights:



2.4.1 Single-Layer Net:-

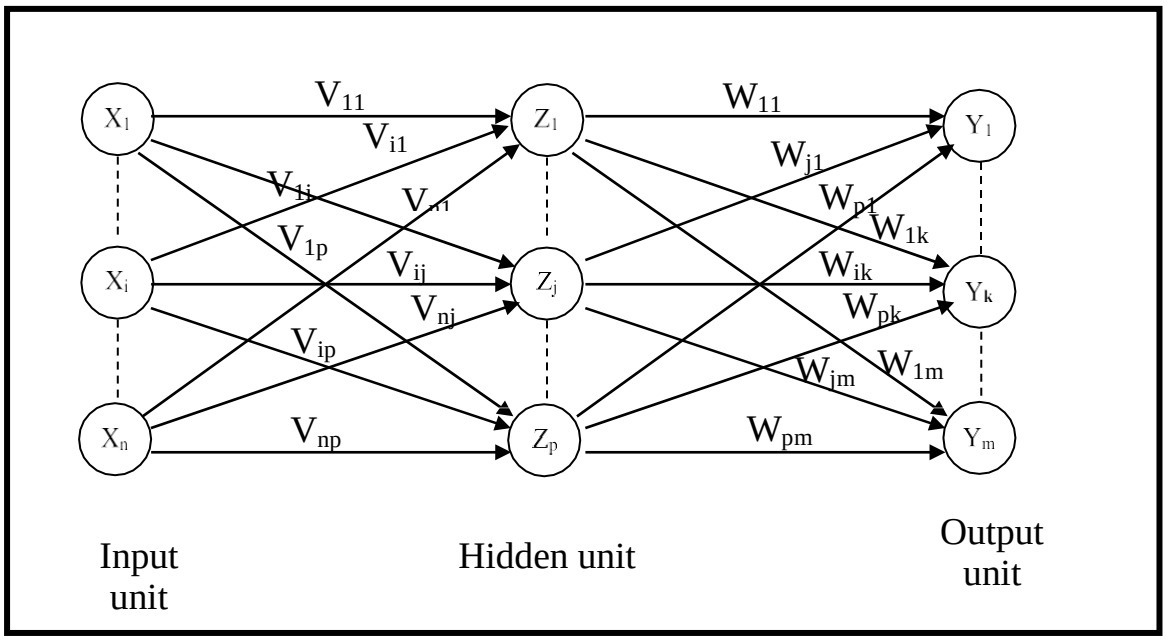
A single-layer net has one layer of connection weight. Often, the units can be distinguished as input units, which receive signals from the outside world, and output units, from which the response of the net can be read. In the typical single-layer net shown in figure bellow the input units are fully connected to output units but are not connected to other input units and the output units are not connected to other output units.



(A single-layer neural network)

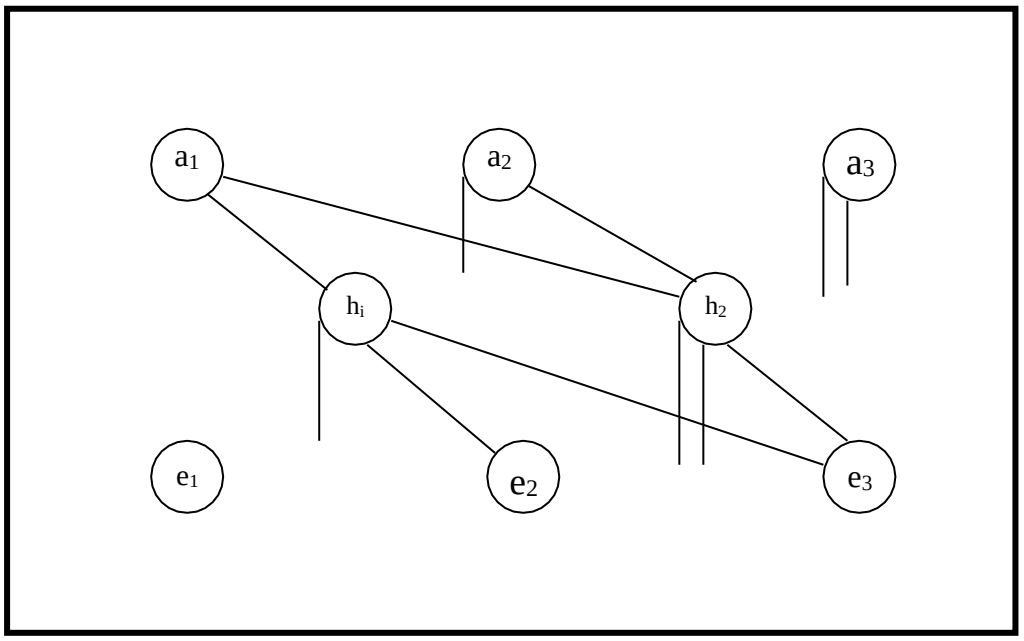
2.4.2 Multilayer net

A Multilayer net is a net with one or more layers (or levels) of nodes which is called hidden units, between the input units and the output units. Typically, there is a layer of weights between two adjacent levels of units (input, hidden, or output). Multilayer nets can solve more complicated problems than can single-layer nets, but training may be more difficult. However, in some cases, training may be more successful because it is possible to solve a problem that a single-layer net can not be trained to perform correctly at all. The figure bellow shows the multilayer neural net.



(A Multilayer Neural Net)

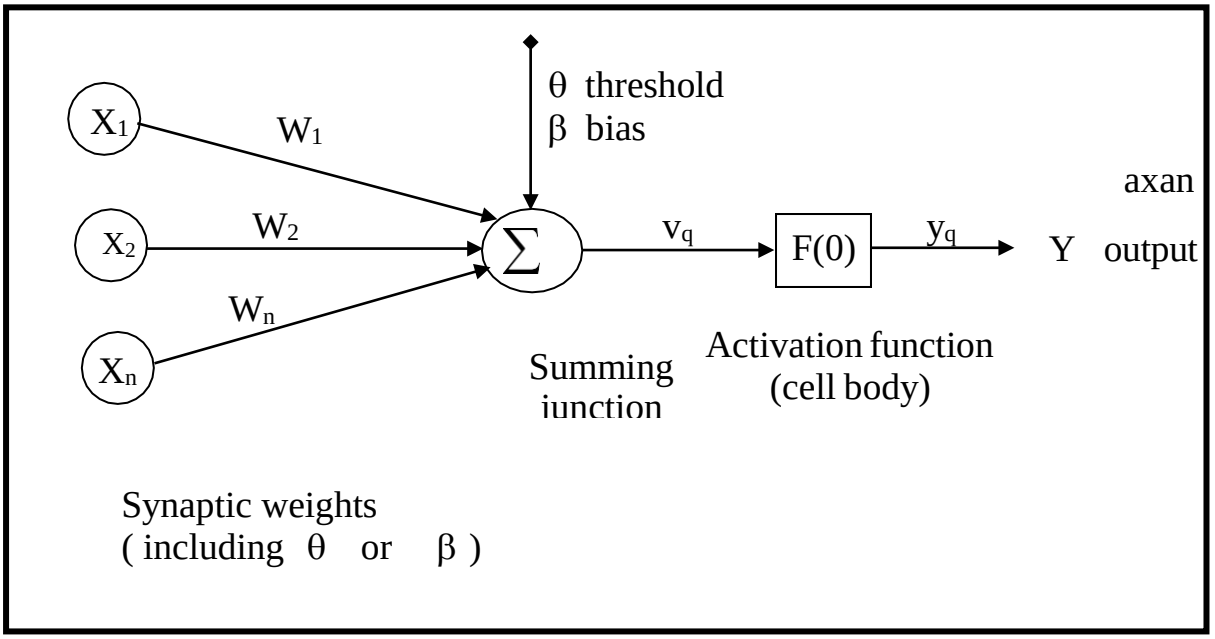
The figure shown bellow is an example of a three-layered neural net work with two hidden neurons.



2.5 Basic Activation Functions

The activation function (Sometimes called a transfers function) shown in figure below can be a linear or nonlinear function. There are many different types of activation functions. Selection of one type over another depends on the particular problem that the neuron (or neural network) is to solve. The most common types of activation function are:-

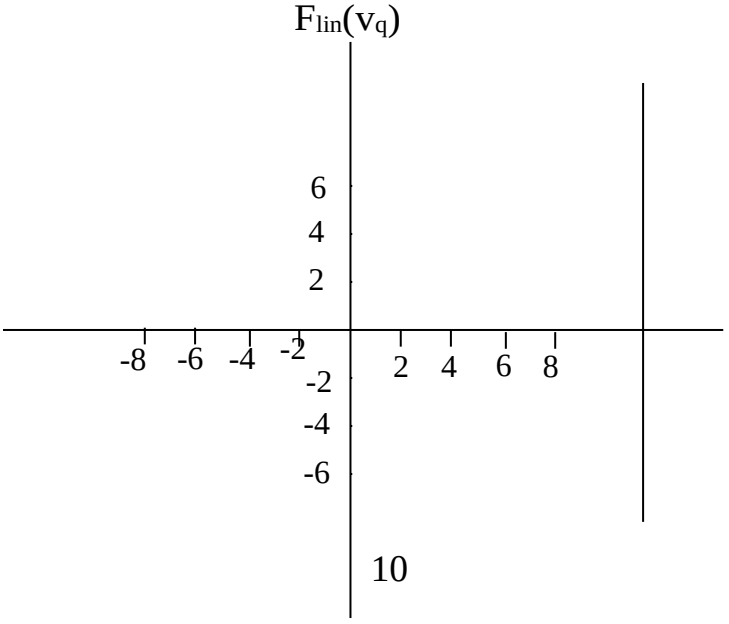
$$V_q = \sum_{v=0}^{n} W_{qj} X_j$$



Alternate nonlinear model of an ANN

1- The first type is ***the linear*** (or ***identity***) function. Ramp

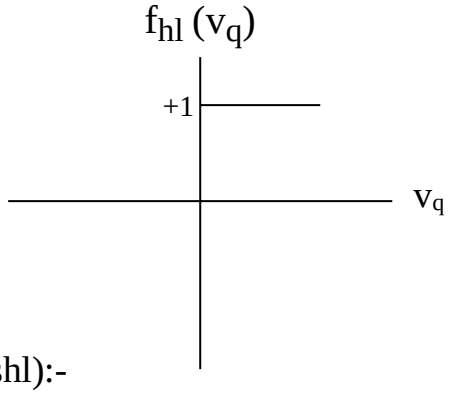
$$y_q f_{lin}(v_q) = v_q$$



2-The second type of activation function is a ***hard limiter***; this is a binary (or bipolar) function that hard-limits the input to the function to either a 0 or a 1 for the binary type, and a -1 or 1 for the bipolar type. The binary hard limiter is sometimes called the threshold function, and the bipolar hard limiter is referred to as the symmetric hard limiter.

a- The o/p of the binary hard limiter:-

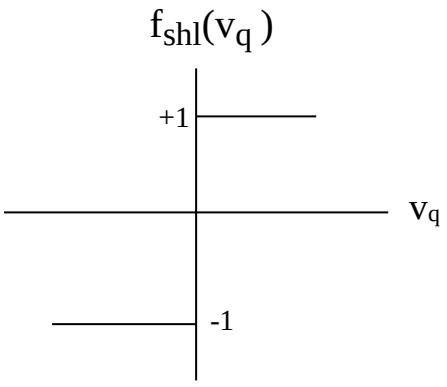
$$y_q = f_{hl}(v_q) = \begin{cases} 0 & \text{if } v_q < 0 \\ 1 & \text{if } v_q >= 0 \end{cases}$$



b-The o/p for the symmetric hard limiter (shl):-

$$y_q = f_{shl}(v_q) = \begin{cases} -1 & \text{if } v_q < 0 \\ 0 & \text{if } v_q = 0 \\ 1 & \text{if } v_q > 0 \end{cases}$$

double side نصرمی اپض



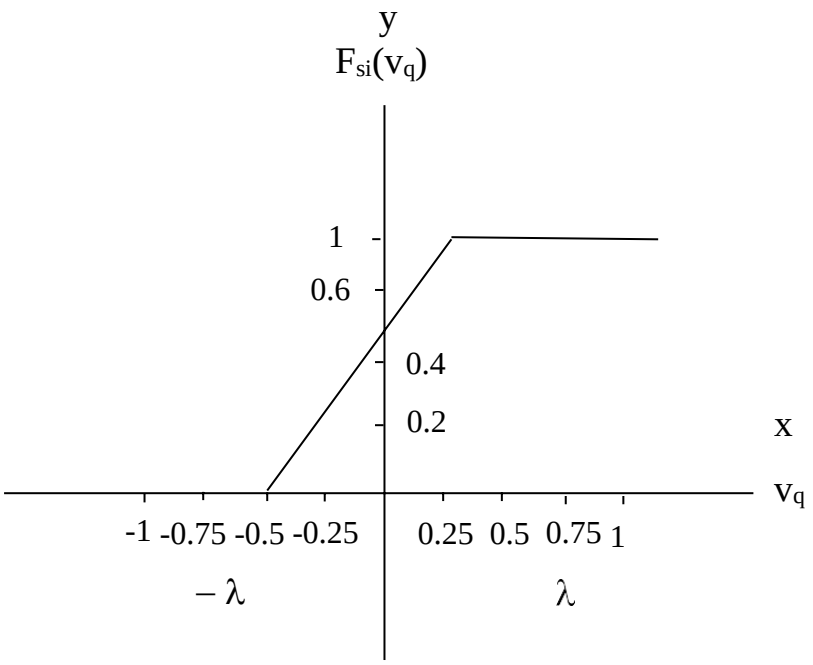
3-The third type of basic activation function is the ***saturating linear*** function or threshold logic Unite (tLu) .

This type of function can have either a binary or bipolar range for the saturation limits of the output. The bipolar saturating linear function will be referred to as the symmetric saturating linear function.

a- The o/p for the ***saturating linear*** function (binary o/p):-

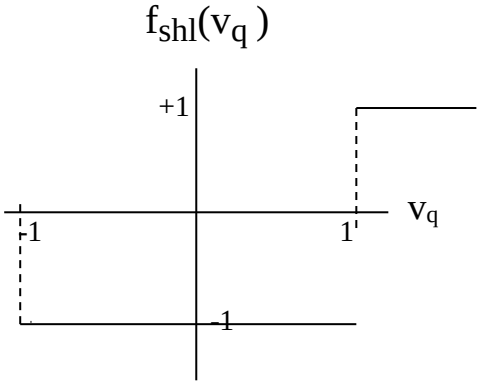
$$y_q = f_{sl}(v_q) = \begin{cases} 0 & \text{if } v_q < -1/2 \\ v_q + 1/2 & \text{if } -1/2 <= v_q <= 1/2 \\ 1 & \text{if } v_q > 1/2 \end{cases}$$

or $$y = \begin{cases} \lambda & \text{if } x > \lambda \\ x & \text{if } -\lambda <= x <= \lambda \\ -\lambda & \text{if } x < -\lambda \end{cases}$$



b- The o/p for the ***symmetric saturating linear*** function:-

$$y_q = f_{ssl}(v_q) = \begin{cases} -1 & \text{if } v_q < -1 \\ v_q & \text{if } -1 <= v_q <= 1 \\ 1 & \text{if } v_q > 1 \end{cases}$$

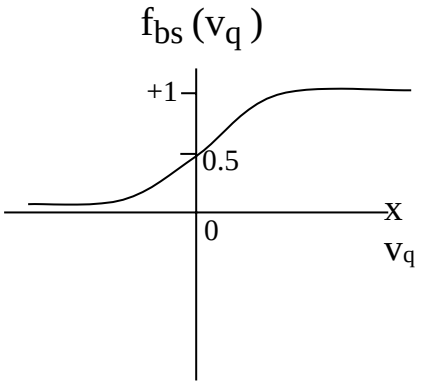


4-The fourth type is ***sigmoid***. Modern NN's use the sigmoid nonlinearity which is also known as logistic, semi linear, or squashing function.

مرحصورة بين 0 و 1وبهذا مرونة

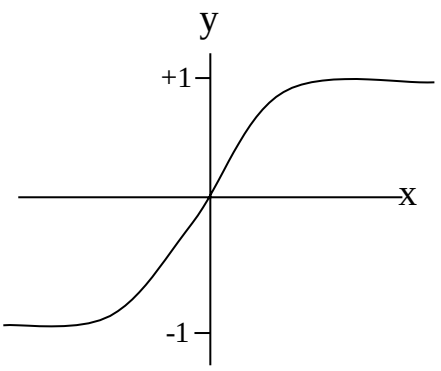
$$y_q = f_{bs}(v_q) = \frac{1}{1+ e^{-\propto v_q}}$$

$$y = \frac{1}{1+ e^{-x}}$$

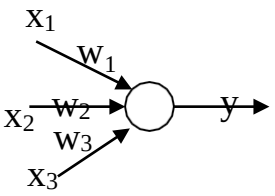


5-***Hyperbollc tangent*** function is similar to sigmoid in shape but symmetric about the origin. (tan h)

$$y = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$



Ex.1 find y for the following neuron if :- x_1=0.5, x_2=1, x_3=0.7
w_1=0, w_2=-0.3, w_3=0.6



Sol

net = = $X_1W_1 + X_2W_2 + X_3W_3$

=0.5*0+1*-0.3+(-0.7*0.6)= -0.72

1- if f is linear

y = -0.72

2- if f is hard limiter (on-off)

y = -1

3-if f is sigmoid

$$y = \frac{1}{1+e^{-(-0.72)}} = 0.32$$

4-if f is tan h

$$y = \frac{e^{-0.72} - e^{0.72}}{e^{-0.72} + e^{+0.72}} = -0.6169$$

5-if f is (TLU) with b=0.6, a=3 then y=-3

$$f(y) = \begin{cases} a & y > b \\ ky & -b < y < b \\ -a & y < -b \end{cases} \quad \Big| \quad f(y) = \begin{cases} a & y > b \\ ky & 0 < y < b \end{cases}$$

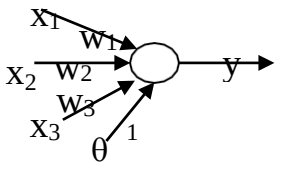
Ex2:- (H.W)

Find y for the following neuron if

$x_1 = 0.5,\ x_2 = 1,\ x_3 = -0.7$

$w_1 = 0,\ w_2 = -0.3,\ w_3 = 0.6$

$\theta = 1$



Sol

Net $= \sum W_i X_i + \theta$

$= -0.72 + 1 = 0.28$

1- if f is linear

$y = 0.28$

2- if f is hard limiter

$y = 1$

3-if f is sigmoid

$$y = \frac{1}{1+e^{-0.28}} = 0.569$$

4-if f is tan sh

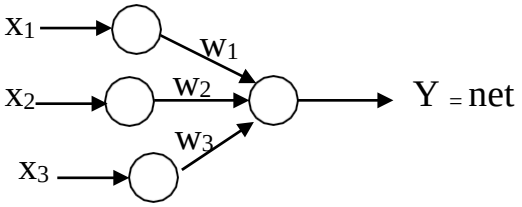
$$y = \frac{e^{0.28} - e^{-0.28}}{e^{0.28} + e^{0.28}} = 0.272$$

5-if f is TLU with b=0.6, +a=3

y=0.28 $y \leftarrow -b < y < b$

Ex.3

The output of a simulated neural using a sigmoid function is 0.5 find the value of threshold when the input $x_1 = 1$, $x_2 = 1.5$, $x_3 = 2.5$. and have initial weights value = 0.2.



Sol

Output = F (net + θ)

$$F(net) = \frac{1}{1+ e^{-net}}$$

$$Net = \sum W_i X_i$$

$$= X_1W_1 + X_2W_2 + X_3W_3$$

$$=(1*0.2)+(1.5*0.2)+(2.5*0.2) = 0.2 +0.30 +0.50 = 1$$

$$0.5 = \frac{1}{1+ e^{-(1+\theta)}}$$

$$0.5\ (1 + e^{-(1+\theta)}) = 1$$

$$0.5+ 0.5\ e^{-(1+\theta)} = 1$$

$$0.5\ e^{-(1+\theta)} = 0.5$$

$$e^{-(1+\theta)} = 1$$

$$-(1+\theta) = \ln 1 \Rightarrow -1 - \theta = \theta \Rightarrow -\theta = 1 \Rightarrow \therefore \theta = -1$$

2.6 The Bias

قيمة ثابتة تضاف لتحسين التعلم

Some networks employ a bias unit as part of every layer except the output layer. This units have a constant activation value of 1 or -1, it's weight might be adjusted during learning. The bias unit provides a constant term in the weighted sum which results in an improvement on the convergence properties of the network.

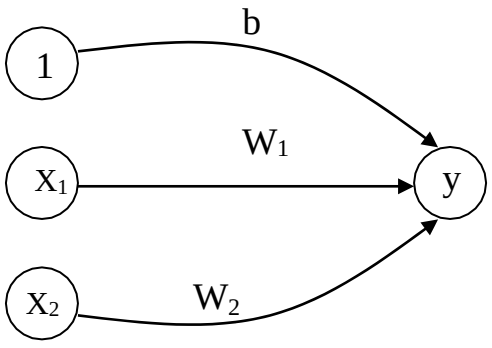
A bias acts exactly as a weight on a connection from a unit whose activation is always 1. Increasing the bias increases the net input to the unit. If a bias is included, the activation function is typically taken to be:

$$f(net)\begin{cases} 1 & \text{if } net >= 0; \\ -1 & \text{if } net < 0; \end{cases}$$

Where

$$net = b + \sum_i X_i W_i$$

Figure: - single –layer NN for logic function



Input unit output unit

Same authors do not use a bias weight, but instead use a fixed threshold θ for the activation function.

$$f(net)\begin{cases} 1 & \text{if } net >= 0; \\ -1 & \text{if } net < 0; \end{cases}$$

Where

$$net = b + \sum_i X_i W_i$$

However, this is essentially equivalent to the use of an adjustable bias.

3.1 Learning Algorithms

The NN's mimic the way that a child learns to identify shapes and colors NN algorithms are able to adapt continuously based on current results to improve performance. Adaptation or learning is an essential feature of NN's in order to handle the new "environments" that are continuously encountered. In contrast to NN's algorithms, traditional statistical techniques are not adoption but typically process all training data simultaneously before being used with new data. The performance of learning procedure depends on many factors such as:-

1- The choice of error function.

2- The net architecture.

3- Types of nodes and possible restrictions on the values of the weights.

4- An activation function.

The convergent of the net:-

Depends on the:-

1- Training set

2- The initial conditions

3- Learning algorithms.

Note:-

The convergence in the case of complete information is better than in the case of incomplete information

Training a NN is to perform weights assignment in a net to minimize the o/p error. The net is said to be trained when convergence is achieved or in other words the weights stop changing.

The learning rules are considered as various types of the:-

3.1.1 Hebbian Learning Rule

The earliest and simplest learning rule for a neural net is generally known as the Hebb rule. Hebbian learning rule suggested by Hebb in 1949. Hebb's basic idea is that if a unit U_j receives an input from a unit U_i and both unite are highly active, then the weight W_{ij} (from unit i to unit j) should be strengthened. This idea is formulated as:-

$$\Delta w_{ij} = \zeta \; x_i y_j$$

Where ζ is the learning rate $\zeta = \propto = 1$, Δw is the weight change

$$w(new) = w(old) + xy$$

$$\therefore \; w(new) = w(old) + \Delta w$$

- محاسن Hebbian learning :-

تسمح بتقليل قيم الـ weight عندما الـ I/p تكون active والـ o/p active وهذه من السهولة بمكان

انجازها باستخدام 1- and +1 بدال من 0 and 1 .

- مساوئ الـ Hebbian learning:-

Hebbian learning takes no account of the actual value of the output, only the desired value. This limitation can be overcome if the weights are adjusted by amount which depends upon the error between the desired and actual output. This error is called delta, S, and the new learning rule is called the delta rule.

Algorithm (Hebbian learning Rule)

Step 0: Initialize all weights

$w_i = 0$ (i =1 to n)

Step 1: for each I/p training vector target o/p

Pair. S : t do steps 2- 4.

Step 2 : Set activations for I/P units:

$w_i = s_i$ (i =1 to n)

Step 3 : set activation for O/P unit :

y = t

Step 4 : Adjust the weights for

w_i (new) = w_i(old) + $x_i y$ (i =1 to n)

Adjust the bias:

b(new) = b(old) + y

Note that the bias is adjusted exactly like a weight from a "unit" whose output signal is always 1.

Ex 4:

A Hebb net for the ABD function: binary input and targets

Input		Target	
1	1	1	1
1	0	1	0
0	1	1	0
0	0	1	0

$\Delta w_1 = x_1 y$, $\Delta w_2 = x_2 y$, $\Delta b = y$

Initial weights = 0, w_1 =0 , w_2 =0, w_3 =0

	x_1	x_2	b	y	Δw_1	Δw_2	Δb	w_1 0	w_2 0	b 0
1	1	1	1	1	1	1	1	1	1	1
2	1	0	1	0	0	0	0	1	1	1
3	0	1	1	0	0	0	0	1	1	1
4	0	0	1	0	0	0	0	1	1	1

The first input pattern shows that the response will be correct presenting the second, third, and fourth training i/p shows that because the target value is 0, no learning occurs. Thus, using binary target values prevents the net from learning only pattern for which the target is "off".

The AND function can be solved if we modify its representation to express the inputs as well as the targets in bipolar form. Bipolar representation of the inputs and targets allows modifications of a weight when the input unit and the target value are both "on" at the same time and when they are both "off" at the same time and all units will learn whenever there is an error in the output. The Hebb net for the AND function: bipolar inputs and targets are:

$\Delta w_1 = x_1 * y$

$=1 * 1 = 1$

$w_1(new) = w_1(old) + \Delta w_1$

$= 0 + 1 = 1$

x_1	x_2	b	y
1	1	1	1
1	-1	1	-1
-1	1	1	-1
-1	-1	1	-1

Presenting the first input:-

x_1	x_2	b	y	Δw_1	Δw_2	Δb	w_1 0	w_2 0	b 0
1	1	1	1	1	1	1	1	1	1

Presenting the second input:-

x_1	x_2	b	y	Δw_1	Δw_2	Δb	w_1 1	w_2 1	b 1
1	-1	1	-1	-1	1	-1	0	2	0

Presenting the third input:-

x_1	x_2	b	y	Δw_1	Δw_2	Δb	w_1 0	w_2 2	b 0
-1	1	1	-1	1	-1	-1	1	1	-1

Presenting the fourth input:-

x_1	x_2	b	y	Δw_1	Δw_2	Δb	w_1 1	w_2 1	b -1
-1	-1	1	-1	1	1	-1	2	2	-2

The first iteration will be:-

Input			target	Weight change			weights		
x_1	x_2	b	y	Δw_1	Δw_2	Δb	w_1 0	w_2 0	b 0
1	1	1	1	1	1	1	1	1	1
1	-1	1	-1	-1	1	-1	0	2	0
-1	1	1	-1	1	-1	-1	1	1	-1
-1	-1	1	-1	1	1	-1	2	2	-2

Second Method

$W_{ij} = \sum X_i Y_j$ ***or*** $[W] = [X]^T [Y]$

Ex. 5

What would the weights be if Hebbian learning is applied to the data shown in the following table? Assume that the weights are all zero at the start.

p	x_1	x_2	y
1	0	0	1
2	0	1	1
3	1	0	0
4	1	1	1

With weights that you've just found, what output values are produce with a threshold of 1, using hyperbolic activation function.

	x_1	x_2	y	Δw_1	Δw	w_1 0	w_2 0
1	0	0	1	0	0	0	0
2	0	1	1	0	1	0	1
3	1	0	0	0	0	0	1
4	1	1	1	1	1	1	2

$p = 0, \quad w_1 = 0, \quad w_1 = 0$

$p = 1, \quad w_1 = 0 + x_1 * y = 0$

$w_2 = 0 + x_2 * y = 0$

$p = 2, \quad w_1 = 0 + 0 * 1 = 0$

$w_2 = 0 + 1 * 1 = 1$

$p = 3, \quad w_1 = 0 + 1 * 0 = 0$

$w_2 = 1 + 0 * 0 = 1$

$p = 4, \quad w_1 = 0 + 1 * 1 = 1$

$w_2 = 1 + 1 * 1 = 2$

$\therefore \quad w_1 = 1, \; w_2 = 2$

$$net = \sum_{i=1}^{4} X_i + W_i$$

$p = 1, \quad net = x_1 * w_1 + x_2 * w_2$

$= 0 * 1 + 0 * 2 = 0$

$p = 2, \quad net = 0 * 1 + 1 * 2 = 2$

$p = 3, \quad net = 1 * 1 + 0 * 2 = 1$

$p = 4, \quad net = 1 * 1 + 1 * 2 = 3$

$$output = F(net + \theta) = \frac{e^{(net+\theta)} - e^{-(net+\theta)}}{e^{(net+\theta)} + e^{-(net+\theta)}}$$

p	x_1	x_2	net	y
1	0	0	0	0
2	0	1	2	1
3	1	0	1	0
4	1	1	3	1

3.1.2 Basic Delta Rule (BDR)

The idea of Hebb was modified to produce the widrow-Hoff (delta) rule in 1960 or least Mean Square (LMS). The BDR is formulated as:-

$$\Delta w_{ij} = \xi(d_j - y_i)\, x_i$$

$$\Delta w_{ij} = \xi\, \delta_j\, x_i \quad \text{(Delta rule)}$$

Δw : - is the weight change

ξ : - is the learning rate

d :- desired output

y :- actual output

δ : - error between d and y

Note:-

Before training the net, a decision has to be made on the setting of the learning rate. Theoretically, the larger ξ the faster training process goes. But practically, ξ may have to be set to a small value (e.g 0.1) in order to prevent the training process from being trapped at local minimum resulting at oscillatory behavior.

H.W

Q1:-Briefly discuss the following:

A-Dendrites

B-synapses

Q2:- A fully connected feed forward network has 10 source nodes, 2 hidden layers, on with 4 neurons and other with 3 neurons, and single output neuron. Construct an architecture graph of this network.

Q3:- A neuron j receives input from four other neurons whose activity levels are 10, -20, 4 and -2. The respective synaptic weights of neuron j are 0.8, 0.2, -1, and -0.9. Calculate the output of neuron j for the following two activation functions:-

i) Hard-limiting function

ii) Logistic function $F(x) = 1/(1+e^{-x})$.

Q4:- perform 2 training steps of the Delta learning rules using $\xi = 1$ & the following data specifying the initial weights W_1, & the two training pairs

$$W_1 = \begin{Bmatrix} 0 \\ 1 \\ 0 \end{Bmatrix}, \begin{Bmatrix} x_1 = \begin{Bmatrix} 2 \\ 1 \\ -1 \end{Bmatrix}, d_1 = -1, x_2 = \begin{Bmatrix} 0 \\ -1 \\ -1 \end{Bmatrix}, d_2 = 1 \end{Bmatrix}$$

Q5:-list the features that distinguish the delta rule & Hebb's rule from each other?

3.1.3 Back Propagation

The determination of the error is a recursive process which start with the o/p units and the error is back propagated to the I/p units. Therefore the rule is called error Back propagation (EBP) or simply Back Propagation (BP). The weight is changed exactly in the same form of the standard DR

$$\Delta w_{ij} = \xi\, \delta_j\, x_i$$

$$\Rightarrow \quad w_{ij}\,(t+1) = w_{ij}\,(t) + \xi\, \delta_j\, x_i$$

There are two other equations that specify the error signal. If a unite is an o/p unit, the error signal is given by:-

$$\delta = (d_j - y_j)\, f_j(\text{net}\ j)$$

$$\text{Where} \quad \text{net}\ j = \sum w_{ij}\, x_i + \theta$$

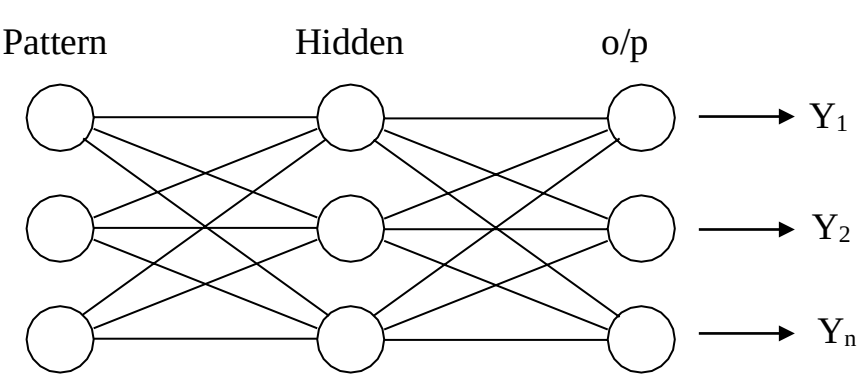
The GDR minimize the squares of the differences between the actual and the desired o/p values summed over the o/p unit and all pairs of I/p and o/p vectors. The rule minimize the overall error $E = \sum E_p$ by implementing a gradient descent in E: - where, $E_p = 1/2 \sum_j (d_j - y_j)^2$.

The BP consists of two phases:-

1- Forward Propagation:-

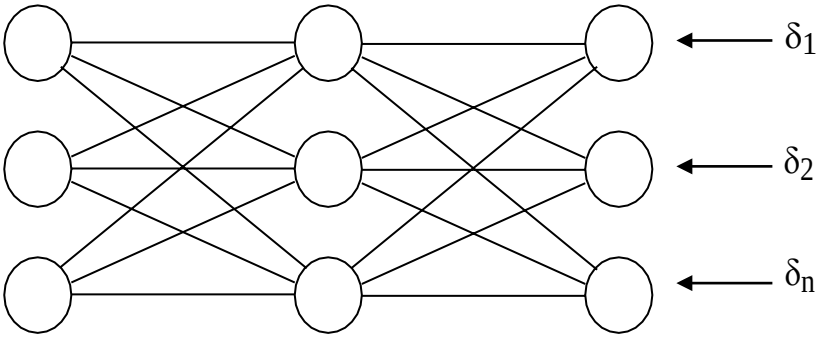
During the forward phase, the I/p is presented and propagated towards the o/p.

المرحلة الأولى



2- Backward Propagation:-

During the backward phase, the ***errors*** are formed at the o/p and propagated towards the I/p



3- Compute the error in the hidden layer.

$$\text{If } y = f(x) = \frac{1}{1+ e^{-x}}$$

$$f' = y(1- y)$$

Equation is can rewrite as:-

$$\delta_j = y(1- y)(d_j - y_j)$$

The error signal for hidden units for which there is no specified target (desired o/p) is determined recursively in terms of the error signals of the units to which it directly connects and the weights of those connections:-

That is

$$\delta_j = f'(net_j)\sum_k \delta_k w_{ik}$$

Or

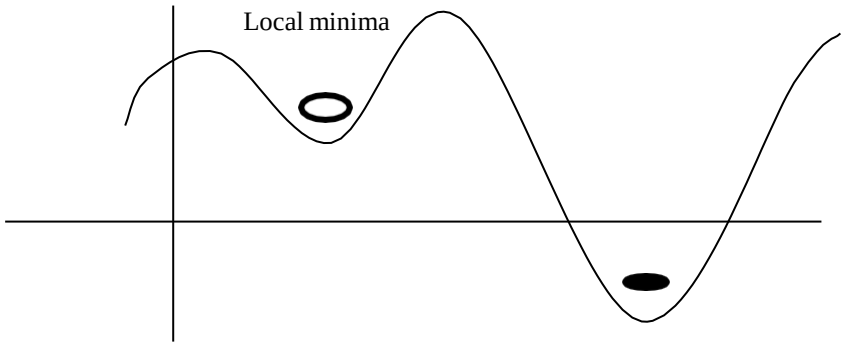
$$\delta_j = y_j(1- y_j)\sum_k \delta_k w_{ik}$$

B.P learning is implemented when hidden units are embedded between input and output units.

Convergence

A quantitative measure of the learning is the :Root Mean Square (RMS) error which is calculated to reflect the "degree" of learning.

Generally, an RMS bellow (0.1) indicates that the net has learned its training set. Note that the net does not provide a yes /no response that is "correct" or "incorrect" since the net get closer to the target value incrementally with each step. It is possible to define a cut off point when the nets o/p is said to match the target values.



- Convergence is not always easy to achieve because sometimes the net gets stuck in a "Local minima" and stops learning algorithm.
- Convergence can be represented intuitively in terms of walking about mountains.

Momentum term

The choice of the learning rate plays important role in the stability of the process. It is possible to choose a learning rate as large as possible without leading to oscillations. This offers the most rapid learning. One way to increase the learning rate without leading to oscillations is to modify the GDR to include momentum term.

This can be achieved by the following rule:-

$$W_{ij}(t+1) = W_{ij}(t) + \zeta\delta_j x_i + \propto (W_{ij}(t) - W_{ij}(t-1))$$

Where $\propto$ ($0 < \propto < 1$) is a constant which determines the effect of the past weight changes on the current direction of movement in weight space.

A "global minima" unfortunately it is possible to encounter a local minima, avally that is not the lowest possible in the entire terrain. The net does not leave a local minima by the standard BP algorithm and special techniques should be used to get out of a local minima such as:-

1- Change the learning rate or the momentum term.
2- Change the no. of hidden units (10%).
3- Add small random value to the weights.
4- Start the learning again with different initial weights.

3.1.3.1 Back Propagation Training Algorithm

Training a network by back propagation involves three stages:-
1-the feed forward of the input training pattern
2-the back propagation of the associated error
3-the adjustment of the weights
let n = number of input units in input layer,
let p = number of hidden units in hidden layer
let m = number of output units in output layer
let V_{ij} be the weights between i/p layer and the hidden layer,
let W_{ij} be the weights between hidden layer and the output layer,
we refer to the i/p units as X_i , i=1, 2, ….,n. and we refer to the hidden units as Z_j , j=1,….,p. and we refer to the o/p units as y_k, k=1,….., m.
δ_{1j} is the error in hidden layer,
δ_{2k} is the error in output layer,
ζ is the learning rate

$\propto$ is the momentum coefficient (learning coefficient, $0.0 < \propto < 1.0$,

y_k is the o/p of the net (o/p layer),

Z_j is the o/p of the hidden layer,

X_i is the o/p of the i/p layer.

η is the learning coefficient.

The algorithm is as following :-

Step 0 : initialize weights (set to small random value).

Step 1 : while stopping condition is false do steps 2-9

Step 2: for each training pair, do steps 3-8

Feed forward :-

Step 3:- Each i/p unit (X_i) receives i/p signal X_i & broad casts this signal to all units in the layer above (the hidden layer)

Step 4:- Each hidden unit (Z_j) sums its weighted i/p signals,

$$Z - inj = Vaj + \sum_{i=1}^{n} x_i v_{ij} \quad (Vaj \text{ is abias})$$

and applies its activation function to compute its output signal (the activation function is the binary sigmoid function),

$Z_j f(Z - inj) = 1 \ / (1 + \exp - (Z - inj))$

and sends this signal to all units in the layer above (the o/p layer).

Step 5:- Each output unit (Yk)sums its weighted i/p signals,

$$y - ink = wok + \sum_{j=1}^{p} Zjwjk \quad (\text{where wok is abias})$$

and applies its activation function to compute its output signal.

$y_k = f(y - ink) = 1/(1 + \exp - (y - ink)$

back propagation of error:-

step 6 : Each output unit (y_k , k= 1 tom) receive a target pattern corresponding to the input training pattern, computes its error information term and calculates its weights correction term used to update W_{jk} later,

$$\delta_{2k} = y_k (1- y_k)*(T_k - y_k),$$

where T_k is the target pattern & k=1 to m .

step 7 : Each hidden unit (Z_j, j= 1 top) computes its error information term and calculates its weight correction term used to update Vij later,

$$\delta_{1j} = Zj* (1- Zj) * \sum_{k=1}^{m} \delta 2k Wjk$$

Update weights and bias :-

step 8: Each output unit (y_k, k =1 tom) updates its bias and weights:

$$Wjk(new) = \eta* \delta 2k * Zj+ \propto *[Wjk(dd)],$$

j= 1 to p

Each hidden unit (Z_j, j= 1 to p) update its bias and weights:

$$Vij(new) = \eta* \delta 1j* Xi+ \propto [vij(dd)],$$

I = 1 to n

Step 9 : Test stopping condition.

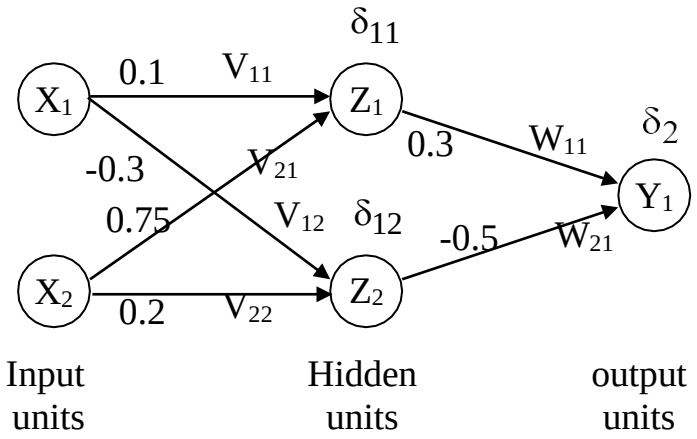
EX6

Suppose you have BP- ANN with 2-input , 2-hiddden , 1-output nodes with sigmoid function and the following matrices weight, trace with 1-iteration.

$$V = \begin{bmatrix} 0.1 & -0.3 \\ 0.75 & 0.2 \end{bmatrix} \qquad w = [0.3 \quad -0.5]$$

Where $\propto = 0.9$, $\eta = 0.45$, x= (1,0), and $T_k = 1$

Solution:-



1-Forword phase :-

$$Z - in1 = X_1V_{11} + X_2V_{21} = 1*\ 0.1 + 0\ *\ 0.75 = 0.1$$
$$Z - in\ 2 = X_1V_{12} + X_2V_{22} = 1*\ -0.3 + 0\ *\ 0.2 = -0.3$$
$$Z_1 = f\ (Z - in1) = 1/(1 + exp-\ (Z - in1)) = 0.5$$
$$Z_2 = f\ (Z - in\ 2) = 1/(1 + exp-\ (Z - in\ 2)) = 0.426$$
$$y - in1 = Z_1W_{11} + Z_2\ W_{21}$$
$$= 0.5*\ 0.3 + 0.426*(-0.5) = -0.063$$
$$y_1 = f\ (y - in1) = 1/(1 + exp-\ (y - in1) = 0.484$$

2-Backward phase :-

$$\delta_2 k = yk(1 - yk) *(Tk - yk)$$

$$\delta_{21} = 0.484(1 - 0.484) * (1 - 0.484)0.129$$

$$\delta_{1j} = Z_j * (1 - Z_j) * \sum_{k=1}^{m} \delta_{2k} W_{jk}$$

$$\delta_{11} = Z_1(1 - Z_1) *(\delta_{21} W_{11})$$
$$= 0.5\ (1- 0.5)*(0.129* 0.3) = 0.0097$$

$$\delta_{12} = Z_2\ (1 - Z_2\ )*(\delta_{21} W_{21})$$
$$= 0.426(1 - 0.426) *(0.129 *(-0.5)) = -0.015$$

3-Update weights:-

$$W_{jk}(new) = \eta * \delta_{2k} * Z_j + \propto *\left[W_{jk}(old)\right]$$

$$W_{11} = \eta * \delta_{21} * Z_1 + \propto * \left[W_{11}(old)\right]$$
$$= 0.45* 0.129* 0.5 + 0.9* 0.3 = 0.299$$

$$W_{21} = \eta * \delta_{21} * Z_2 + \propto * \left[W_{21}(old)\right]$$
$$= 0.45* 0.129* 0.426 + 0.9*-0.5 = -0.4253$$

$$V_{ij}(new) = \eta * \delta_{1j} * X_i + \propto * \left[V_{ij}(old)\right]$$

$$V_{11} = \eta * \delta_{11} * X_1 + \propto * \left[V_{11}(old)\right]$$
$$= 0.45* 0.0097 *1 + 0.9* 0.1= 0.0944$$

$$V_{12} = \eta * \delta_{12} * X_1 + \propto * \left[V_{12}(old)\right]$$
$$= 0.45* 0.0158*1 + 0.9*-0.3= -0.2771$$

$$V_{21} = \eta * \delta_{11} * X_2 + \propto * \left[V_{21}(old)\right]$$
$$= 0.45* 0.0097 * 0 + 0.9* 0.75 = 0.675$$

$$V_{22} = \eta * \delta_{12} * X_2 + \propto * \left[V_{22}(old)\right]$$
$$= 0.45*-0.0158* 0 + 0.9* 0.2 = 0.18$$

$$\therefore V = \begin{bmatrix} 0.0944 & -0.2771 \\ 0.675 & 0.18 \end{bmatrix} \qquad W = \begin{bmatrix} 0.299 & -0.4253 \end{bmatrix}$$

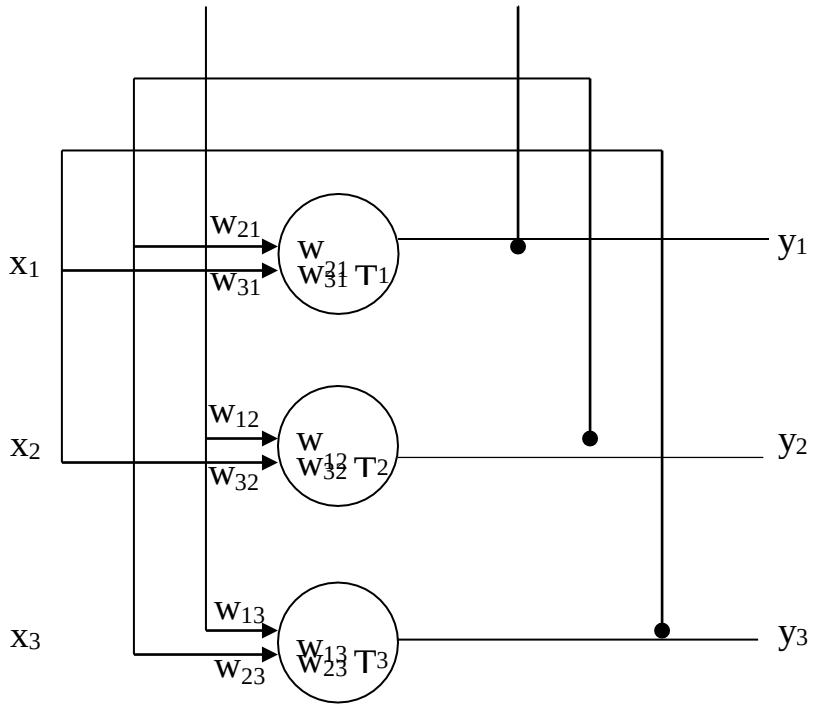
3.2 The Hopfield Network

The Nobel prize winner (in physics) John Hopfield has developed the discrete Hopfield net in (1982-1984). The net is a fully interconnected neural net, in the sense that each unit is connected to every other unit. The discrete Hopfield net has symmetric weights with no self-connections, i.e,

$$W_{ij} = W_{ji}$$

$$\text{And} \quad W_{ii} = 0$$

In this NN, inputs of 0 or 1 are usually used, but the weights are initially calculated after converting the inputs to -1 or +1 respectively.



"The Hopfield network"

The outputs of the Hopfield are connected to the inputs as shown in Figure, Thus feedback has been introduced into the network. The present output pattern is no longer solely dependent on the present inputs, but is also dependent on the previous outputs. Therefore the network can be said to have some sort of memory, also the Hopfield network has only one layer of neurons.

The response of an individual neuron in the network is given by :-

$$y_j = 1 \quad \text{if} \quad \sum_{i=1\ i\neq j}^{n} W_{ij}X_i > T_j$$

$$y_j = 0 \quad \text{if} \quad \sum_{i=1\ i\neq j}^{n} W_{ij}X_i < T_j$$

This means that for the jth neuron, the inputs from all other neurons are weighted and summed.

Note $i \neq j$, which means that the output of each neuron is connected to the input of every other neuron, but not to itself. The output is a hard-limiter which gives a 1 output if the weighted sum is greater than T_j and an output of 0 if the weighted sum is less than T_j. it will be assumed that the output does not change when the weighted sum is equal to T_j.

Thresholds also need to be calculated. This could be included in the matrix by assuming that there is an additional neuron, called neuron 0, which is permanently stuck at 1. All other neurons have input connections to this neuron's output with weight W01, W02, W03,…etc. this provides an offset which is added to the weighted sum. The relation ship between the offset and the threshold T_j is therefore:- Tj = -W0j

The output [y] is just the output of neuron 0 which is permanently stuck at 1, so the formula becomes:- $[W_0] = [X]_t[Y_0]$

For example, if the patterns $X_1 = [0011]$ and $X_2 = [0101]$ are to be stored, first convert them to

$$X_1 = [-1 \quad -1 \quad 1 \quad 1]$$

$$X_2 = [-1 \quad 1 \quad -1 \quad 1]$$

To find the threshold:-

1- The matrix $\begin{bmatrix} -1 & -1 & 1 & 1 \\ -1 & 1 & -1 & 1 \end{bmatrix}$

2-The transpose of the matrix is $\begin{bmatrix} -1 & -1 \\ -1 & 1 \\ 1 & -1 \\ 1 & 1 \end{bmatrix}$

3- y_0 is permanently stuck at +1 , so the offsets are calculated as follows

$$W_0 = \begin{bmatrix} -1 & -1 \\ -1 & +1 \\ +1 & -1 \\ +1 & +1 \end{bmatrix} \begin{bmatrix} +1 \\ \\ +1 \end{bmatrix} = \begin{bmatrix} -2 \\ 0 \\ 0 \\ +2 \end{bmatrix}$$

4-These weights could be converted to thresholds to give:-

$T_1 = 2$
$T_2 = 0$ Tj =-W0j
$T_3 = 0$
$T_4 = -2$

EX7:-

Consider the following samples are stored in a net:-

$$\begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} -1 & +1 & -1 & -1 \\ +1 & +1 & -1 & -1 \\ -1 & -1 & +1 & +1 \end{bmatrix}$$

binary $\rightarrow$ convert $\rightarrow$ bipolar

The binary input is (1110). We want the net to know which of samples is the i/p near to?

Note :-

A binary Hopfield net can be used to determine whether an input vector is a "known" vector (i.e., one that was stored in the net) or "unknown" vector.

Solution:- 1-use Hebb rule to find the weights matrix

$$W = \begin{bmatrix} W_{11} & W_{12} & W_{13} & W_{14} \\ W_{21} & W_{22} & W_{23} & W_{24} \\ W_{31} & W_{32} & W_{33} & W_{34} \\ W_{41} & W_{42} & W_{43} & W_{44} \end{bmatrix}$$

$W_{ii}=0$ (diagonal)

$W_{ij}=W_{ji}$

$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & W_{12} & W_{13} & W_{14} \\ W_{21} & 0 & W_{23} & W_{24} \\ W_{31} & W_{32} & 0 & W_{34} \\ W_{41} & W_{42} & W_{43} & 0 \end{bmatrix} \end{array}$$

$$W_{12} = (-1*1) + (1*1) + (-1*-1) = 1$$
$$W_{13} = (-1*-1) + (1*-1) + (-1*1) = -1$$
$$W_{14} = (-1*-1) + (1*-1) + (-1*1) = -1$$

$$W_{21} = W_{12} = 1$$
$$W_{23} = (1*-1) + (1*-1) + (-1*1) = -3$$
$$W_{24} = (-1*-1) + (1*-1) + (-1*1) = -3$$

$$W_{31} = W_{32} = 1$$
$$W_{32} = W_{23} = -3$$
$$W_{34} = (-1*-1) + (-1*-1) + (1*1) = 3$$

$$W_{41} = W_{14} = -1$$
$$W_{42} = W_{24} = -3$$
$$W_{43} = W_{34} = 3$$

$$\therefore W = \begin{bmatrix} 0 & 1 & -1 & -1 \\ 1 & 0 & -3 & -3 \\ -1 & -3 & 0 & 3 \\ -1 & -3 & 3 & 0 \end{bmatrix}$$

2-The i/p vector x = (1 1 1 0). For this vector, y= (1 1 1 0)

Choose unit y_1 to update its activation

$$y-in1 = X_1 + \sum_{j}^{m} y_j w_{j1}$$

$$y-in1 = 1 + [(0*1) + (1*1) + (-1*1) + (-1*0)]$$

$$=1+0=1$$

$$\therefore y = (1110)$$

Choose unit y_2 to up date its activation:-

$$y-in\,2 = x_2 + \sum_{j} y_j w_{j2}$$

$$= 1 + [(1*1) + (1*0) + (1*-3) + (0*-3)]$$

$$= 1 + (-2) = -1$$

$$y-in\,2 < 0 \qquad \therefore y_2 = 0$$

$$\therefore y = (1010)$$

Choose unit y_3 to update its activation:-

$$y-in\,3 = x_3 + \sum_{j} y_j w_{j3}$$

$$= 1 + [(1*-1) + (1*-3) + (1*0) + (0*3)]$$

$$= 1 + (-4) = -3$$

$$y-in\,3 < 0 \qquad \therefore y_3 = 0$$

$$\therefore y = (1000)$$

Choose unit y_4 to update its activation:-

$$y-in4 = x_4 + \sum_{j} y_j w_{j4}$$

$$= 0+ [(1*-1) + (1*-3) + (1*3) + (0*0)]$$

$$= 0+ (-1) \quad = -1$$

$y\text{-}in4 < 0$ $\quad y4=0$

$y = (1000)$

3- Test for convergence, false

$\therefore$ The input vector x = (1000), for this vector,

Y= (1 0 0 0)

$y - in1 = 1$

$y - in\ 2 = 1$

$y - in3 = -1 = 0$

$y - in4 = -1 = 0$

$\therefore\ y = (1100)$

$\therefore$ The input vector x= (1 1 0 0)

Y= (1 1 0 0)

$y - in1 = 2 = 1$

$y - in\ 2 = 2 = 1$

$y - in3 = -4 = 0$

$y - in4 = -4 = 0$

$\therefore\ y = (1100)$

The input is near to the second sample.

True.

Stop.

H.W

1-find the weights and thresholds for a Hopfield network that stores the patterns:- (0 0 1) and (0 1 1).

2-There are special techniques should be used to get out of local minima, explain it.

3.3 Bidirectional Associative Memory (BAM)

A bidirectional associative memory (BAM) is very similar to a Hopfield network, but has two layers of neurons (kosko, 1988) and is fully connected from each layer to the other. There are feedback connections from the output layer to the input layer.

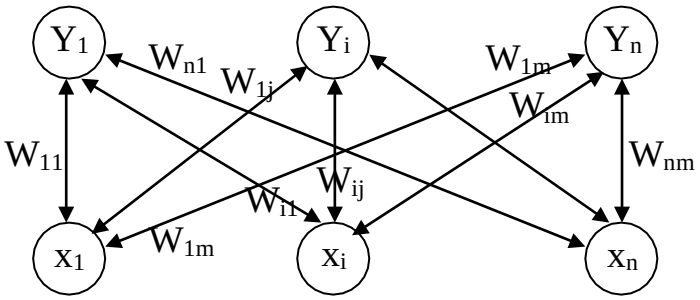
The BAM is hetero associative, that is, it accept on input vector on one set of neurons and produces a related, but different, output vector on another set. The weights on the connections between any two given neurons from different layers are the same.

The matrix of weights for the connections from the output layer to the input layer is simply the transpose of the matrix of weights for the connections between the input and output layer.

Matrix for forward connection weights = w

Matrix for backward connection weights = w^T

There are 2 layers of neurons, an input layer and on output layer. There are no lateral connections, that is, no two neurons within the some layer are connected, Recurrent connections, which are feedback connections to a neuron from itself, may or not be present. Unlike the Hopfield net work, the diagonal of the connection matrix is left intact, also the number of bits in the input pattern need not be the same as the output pattern, so the connection matrix is not necessarily sequare.



“ Layout of BAM Network “

The BAM operates by presenting on input pattern,[A], and passing it through the connection matrix to produce an output pattern,[B] .so:-

$$B[k] = f([A(k)][w])$$

Where

K: indicates time

A(k), B(k) :- are equivalent to [x] and [y]

F: activation function

W:- weight matrix between layer 1 & layer 2

The output of the neurons are produced by the function f( ) which, like the Hopfield, is a hard-limiter with special case at θ .

This output function is defined as follows :-

outi $(k+1) = 1$ if $Neti(k) > 0$

outi $(k+1) = 0$ if $Neti(k) < 0$

outi $(k+1) = outi(k)$ if $Neti = 0$ unchanged

The output [B], is then passed back through the connection matrix to produce a new input pattern, [A].

$$A(k+1) = f([B(k)][W^T])$$

The [A] & [B] pattern are passed back and forth through the connection matrix in the way just described unitl there are no further changes to the values of [A] & [B]

محاسن الـBAM -

1منسجمة مع الموائر النظرية واالنظمة البصرية 2- لها

ائتراب سريع في عملية التعلم واالسترجاع

3-لها حصانة ضم ال- noisy data

4االوزا ثابتة **مساوئ الـ- _**

BAM 1سعة الخز

محمدة 2-لها استجابة

زائفة

3-احياناً نسلك سلوك غير متوقع

No learning -4

EX8:- let us try to train a network to remember three binary – vector pairs. Ai, Bi have the same number of component, using the Hebb rule to star :-

$$A_1 = (1\ 0\ 0) \qquad B_1 = (0\ 0\ 1)$$
$$A_2 = (0\ 1\ 0) \qquad B_2 = (0\ 1\ 0)$$
$$A_3 = (0\ 0\ 1) \qquad B_3 = (1\ 0\ 0)$$

1- Find the weight matrix?

2-Apply an input vector $A_1 = (1\ 0\ 0)$ to test the net to remember A_1.

Sol

$$W_1 = [A_1^T][B_1]$$

$$W_1 = \begin{bmatrix} 1 \\ -1 \\ -1 \end{bmatrix} [-1\ \ -1\ \ 1] = \begin{bmatrix} -1 & -1 & 1 \\ 1 & 1 & -1 \\ 1 & 1 & -1 \end{bmatrix}$$

$$W_2 = [A_2^T][B_2]$$

$$W_2 = \begin{bmatrix} -1 \\ 1 \\ -1 \end{bmatrix} [-1\ \ 1\ \ -1] = \begin{bmatrix} 1 & -1 & 1 \\ -1 & 1 & -1 \\ 1 & -1 & 1 \end{bmatrix}$$

$$W_3 = [A_3^T][B_3]$$

$$W_3 = \begin{bmatrix} -1 \\ -1 \\ 1 \end{bmatrix} [1\ \ -1\ \ -1] = \begin{bmatrix} -1 & 1 & 1 \\ -1 & 1 & 1 \\ 1 & -1 & -1 \end{bmatrix}$$

$$W = W_1 + W_2 + W_3$$

$$W = \begin{bmatrix} -1 & -1 & 1 \\ 1 & 1 & -1 \\ 1 & 1 & -1 \end{bmatrix} + \begin{bmatrix} 1 & -1 & 1 \\ -1 & 1 & -1 \\ 1 & -1 & 1 \end{bmatrix} + \begin{bmatrix} -1 & 1 & 1 \\ -1 & 1 & 1 \\ 1 & -1 & -1 \end{bmatrix} = \begin{bmatrix} -1 & -1 & 3 \\ -1 & 3 & -1 \\ 3 & -1 & -1 \end{bmatrix}$$

Test for A_1

$A_1 * W = B_1$

$$\begin{bmatrix}1 & -1 & -1\end{bmatrix}\begin{bmatrix}-1 & -1 & 3\\ -1 & 3 & -1\\ 3 & -1 & -1\end{bmatrix} = \begin{bmatrix}-3 & -3 & 5\end{bmatrix}$$

$$\begin{bmatrix}-3 & -3 & 5\end{bmatrix} = \begin{bmatrix}0 & 0 & 1\end{bmatrix} = B_1$$

Or طريقة مختصرة

$W = [AT]\,[B]$

$$W = \begin{bmatrix}1 & -1 & -1\\ -1 & 1 & -1\\ -1 & -1 & 1\end{bmatrix}\begin{bmatrix}-1 & -1 & 1\\ -1 & 1 & -1\\ 1 & -1 & -1\end{bmatrix} = \begin{bmatrix}-1 & -1 & 3\\ -1 & 3 & -1\\ 3 & -1 & -1\end{bmatrix}$$

And then continues the same steps .

H.W

Q1: find the weights and thresholds for a Hopfield network that stores the pattern 001 and 011.

Q2- A BAM is trained using the following input and output patterns:-

Input	Output
000010010000010	01
000010000010000	10
000100100100000	11

Find the weights that would be generated for the BAM network, and check that the input patterns generate the corresponding output patterns.

Q3- Briefly explain the following :-

1- Single layer network , Multi layer network

2-ANN

3- Areas of Neural network

4- supervised Learning , unsupervised Learning

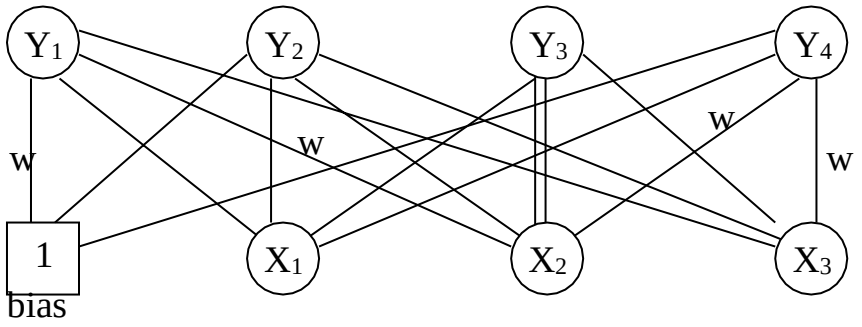
5-Recurrnt, non recurrent

6-Advantage & disadvantage of BAM

7- write the complete alg. Of BAM.

3.4 Adaline Neural Network

Adaline is the short from of "Adaptive linear neuron" and was presented in 1960 by B. Widrow and N. E. Hoff [WH1960]. The network is single layered, the binary values to be assumed for input and output are -1 and +1 respectively. Figure bellow shows the general topology of the network.



"Topology of Adaline "

Where

X = input vector (including bias)

Y=output vector = f(w*x)

W=weight matrix

An Adaline can be trained using the delta rule, also known as the least mean sequares (LMS) or widerow- Holf rule. The learning rule minimize the mean squared error between the activation and the target value. This allows the net to continue learning on all training patterns, even after the correct output value is generated (if a threshold function is applied) for some patterns.

When the Adaline is in its tracing or learning phase, there are three factors to be taken into account

1-the inputs that are applied are chosen from a training set where the desired response of the system to these inputs is known.

2-the actual output produced when an input pattern is applied is compared with the desired output and used to calculate an error δ

3- the weight are adjusted to reduce the error .

This kind of training is called supervised learning because the output are known and the network is being forced into producing the correct outputs.

Three additional points need to be included before the learning rule can be used:-

4-the constant, η , has to be decided. The original suggestion for the Adaline was that η is made equal to:-

$\eta = 1/(n+1)$

Where n is the number of inputs.

The effect of adjusting the weights by this amount is to reduce the error for the current input pattern to zero. In practice if η is sat to this value the weights rarely settle down to a constant value and a smaller value is generally used.

5-the weight are initially set to a small random value. This is to ensure that the weights are all different.

6-the offset, w_0 gets adjusted in the same way as the other weights, except that the corresponding input x_0 is assumed to be +1.

The steps for solving any question in Adaline by using Delta-rule are :-

1-compute the learning coefficient η :-

$$\eta = 1/(n+1)$$

n= number of inputs

2-comput neti :-

$$\text{neti} = \sum_{i=1} x_i . w_i$$

3-compute the error δ

$$\delta = d - \text{neti} \qquad d \text{ is the desired o/p}$$

4-compute the value of $\eta \cdot \delta \cdot x_i$ for all weights

5-find the total for all weight $\text{total} = \sum \eta \delta x_i$

6-find mean i mean i = total /p

Where :-

P:- is the no. of states

7- adjust the weights depending on meani $W_i^{new} = W_i^{old} + meani$

EX9 :-

Adaline is given the four different input and output combinations of the two input AND function, $y = x_1 \wedge \neg x_2$, as training set

$w_0 = -0.12$

$w_1 = 0.4$

$w_2 = 0.65$

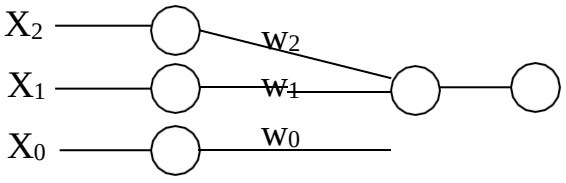
$y = x_1 \wedge \neg x_2$

X_1	X_2	Y
0	0	0
0	1	0
1	0	1
1	1	0

bias

X_0	X_1	X_2	Y
+1	-1	-1	-1
+1	-1	+1	-1
+1	+1	-1	+1
+1	+1	+1	-1

Desired o/p



First the input pattern : +1 -1 -1

Weights : -0.12 0.4 0.65

$$net = \sum_{i=1}^{n} x_i . w_i$$

= (+1*-0.12)+(-1*0.4) +(-1*0.65) = -1.17 (actual output)

d = desired output = -1 (for first pattern)

$\therefore \delta = d - net$

= -1-(-1.17) =0.17 (error)

$\eta = 0.1$

Also we must compute :-

$\Delta w_{ij} = \eta \cdot \delta \cdot x_i$

For convenience , these figures have been rounded to two places after the decimal point, so become :- $\eta \cdot \delta \cdot x_0 = (0.1 * 0.17 * +1) = 0.017 \approx = 0.02$

ملاحظة:- نلاحظ بالعمل مع بقية ال input وبعد الحصول على النتائج النهائية:-

								Δw_0	Δw_1	Δw_2
X_0	X_1	X_2	W_0	W_1	W_2	net	d	$\eta\delta x_0$	$\eta\delta x_1$	$\eta\delta x_2$
+1	-1	-1	-0.12 -	0.40	0.65	-1.17	-1	0.02	-0.02	-0.02
+1	-1	+1	0.12	0.40	0.65	0.13	-1	-0.11	0.11	-0.11
+1	+1	-1	-0.12	0.40	0.65	-0.37	+1	0.14	0.14	-0.14
+1	+1	+1	-0.12	0.40	0.65	0.93	-1	-0.19	-0.19	-0.19
							total	-0.14	0.04	-0.46

$meanj = total(\Delta wij)/p \qquad p = 4$

$Mean_0 = -0.14/4 = -0.035 = -0.04$

$Mean_1 = -0.04/4 = -0.01$

$Mean_2 = -0.46/4 = -0.115 = -0.12$

$\therefore W_{ij}^{new} = W_{ij}^{old} + meanj$

$W_0^{new} = -0.12 + (-0.04) = -0.16$

$W_1^{new} = -0.40 + (0.01) = -0.41$

$W_2^{new} = -0.66 + (-0.12) = 0.53$,

Continue until $\eta\delta x = 0$

X_0	X_1	X_2	W_0	W_1	W_2	net	d	$\eta\delta x_0$	$\eta\delta x_1$	$\eta\delta x_2$
+1	-1	-1	-0.16	0.41	0.53	-1.10	-1	0.01	-0.01	-0.01
+1	-1	+1	-0.16	0.41	0.53	0.04	-1	-0.10	0.10	-0.10
+1	+1	-1	-0.16	0.41	0.53	-0.25	+1	0.13	0.13	-0.13
+1	+1	+1	-0.16	0.41	0.53	0.78	-1	-0.18	-0.18	-0.18
							total	-0.14	0.04	-0.44
							mean	-0.04	0.01	-0.11

								Δw_0	Δw_1	Δw_2
X_0	X_1	X_2	W_0	W_1	W_2	net	d	$\eta\delta x_0$	$\eta\delta x_1$	$\eta\delta x_2$
+1	-1	-1	-0.50	0.50	-0.50	-0.50	-1	-0.05	0.05	0.05
+1	-1	+1	-0.50	0.50	-0.50	-1.50	-1	0.05	-0.05	0.05
+1	+1	-1	-0.50	0.50	-0.50	0.50	+1	0.05	0.05	-0.5
+1	+1	+1	-0.50	0.50	-0.50	-0.50	-1	-0.05	-0.05	-0.5
							total	0.00	0.00	0.00

The network has successfully found a set of weight that produces the correct outputs for all of the patterns.

H.W

Q1:A 2-input Adaline has the following set of weights w_0 =0.3 , w_1=-2.0 ,w_2 = 1.5 When the input pattern is x_0 = 1 , x_1= 1 , x_2 = -1

And the desired output is 1

a- what is the actual output?

b- what is the value of δ?

c- Assuming that the weights are updated after each pattern and the value η is 1/n+1 , what are the new values for the weights?

d- using these new values of weights, what would the output be for the same input pattern?

Q2: with η set to 0.5, calculated the weights (to one decimal place) in the following example after are iteration through the set of training patterns.

a- updating after all the patterns are presented

b- updating after each pattern is presented

X_0	X_1	X_2	W_0	W_1	W_2	net	d	$\eta\delta x_0$	$\eta\delta x_1$	$\eta\delta x_2$
+1	-1	-1	-0.2	0.1	0.3	-0.6	+1	0.8	-0.8	-0.8
+1	-1	+1					+1			
+1	+1	-1					-1			
+1	+1	+1					+1			

3.5 Kohonen Network

Teuvo kohonen presented the self-organizing feature map in 1982. it is an unsupervised, competitive learning , clustering network in which only one neuron (or only one neuron in a group) is "on" at a time.

The self-organizing neural networks, also called (topology –preserving maps), assume a topological structure among the cluster units. This property is observed in the brain, but is not found in other artificial neural networks.

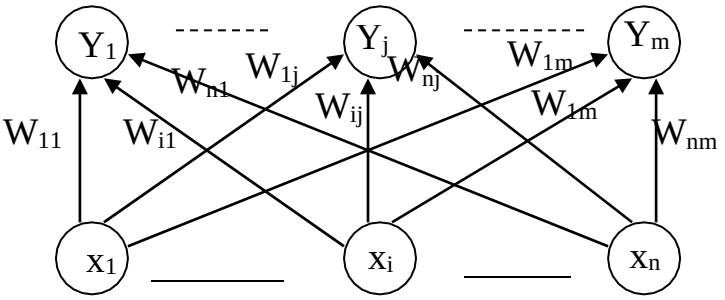
There are m cluster units arranged in a one –or two – dimensional array.

Cluster: وهي مجاميع من المعلومات كل مجموعة لها صفة معينة.

The weight vector for cluster units serves as an exemplar of the input patterns associated with that cluster. During the self organizing process, the cluster unit whose weight vector matches the input pattern most closely (typically, the square of the minimum Euclidean distance) is chosen as the winner. The winning unit and its neighboring units update their weights. The weight vectors of neighboring units are not, in general, close to the input pattern.

3.5.1 Architecture

A kohonen network has two layers, an input layer to receive the input and an output layer. Neurons in the output layer are usually arranged into a regular two dimensional array. The architecture of the kohonen self-organizing map is shown bellow.

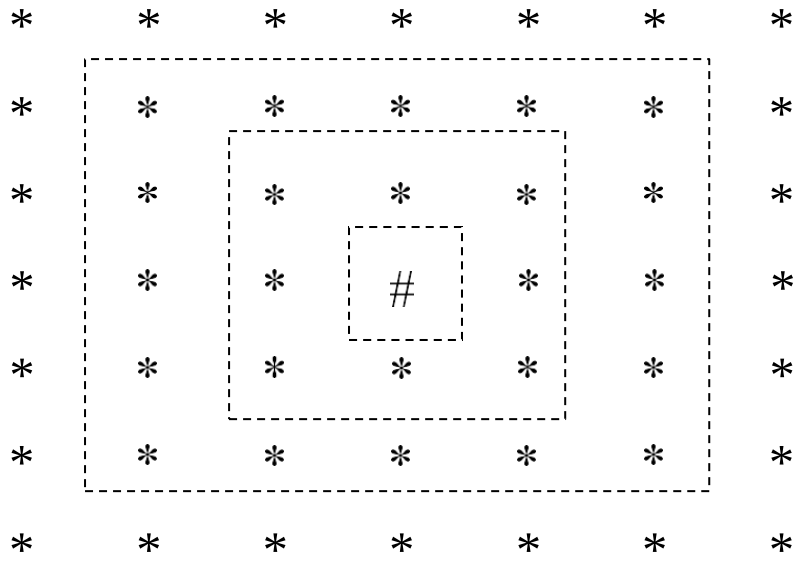


(kohonen self-organizing map)

* * * [* (*[#] *) *] * *
R_2 R_1 R_0 R_1 R_2

Linear array 10 cluster

Neighborhoods of the unit designated by # of radii R=2 (1& 0) in a one – dimensional topology (with 10 cluster units) are shown in figure(4.2)



Neighborhoods for rectangular grid

R_0 = ……..

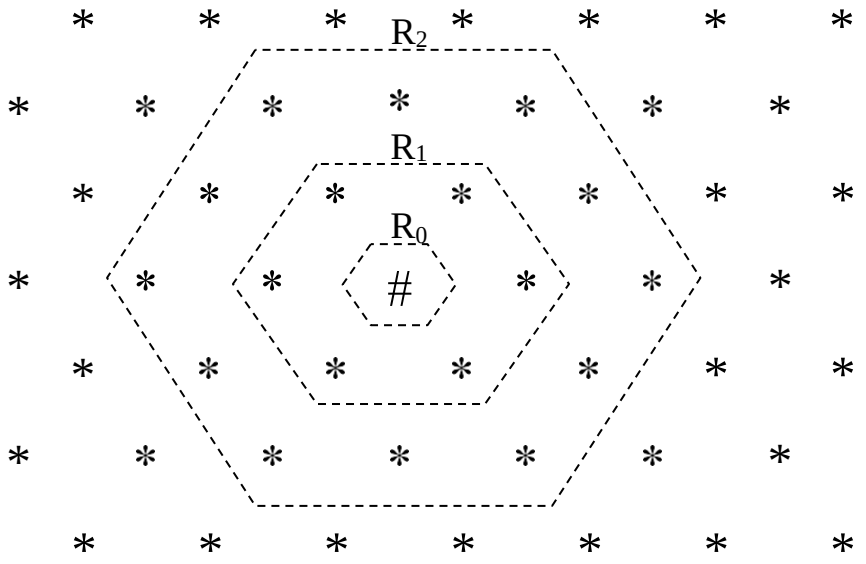
R_1 =

R_2 = - - - - -

* * * * * * * *

The Neighborhoods of unit radii R=2 (1 & 0) are shown in figure (4.3) for a rectangular grid and in figure (4.4) for hexagonal grid (each with 49 units). In each illustration, the winning unit is indicated by the symbol "#" and the other units are denoted by "*" .

* Kohonen NN can be used in speech recognizer



Neighborhoods for hexagonal grid

R_0 = ……..

R_1 =

R_2 = - - - - -

3.5.2 Algorithm

Step 0 : initialize weights wij

Set topological neighborhood parameters

Set Learning rate parameters.

Step1:while stopping condition is false, do step 2-8

Step2: for each input vector x, do step 3-5

Step3: for each j, compute distance

$D(j) = \sum_i (x_i - w_{ij})^2$ Euclidean distances

Step4 : find index J such that D(J) is a minimum

Step5: for all units j within a specified neighborhood of J, and for all i:

Wij(new) = Wij(old)+ $\propto$ [Xi + Wij(old)]

Step6: update learning rate.

Step7: Reduce radius of topological neighborhood at specified times

Step8: Test stopping condition.

EX 10

A kohonen self-organizing map (SOM) to be cluster four vectors

vector1 = (1 1 0 0)
vector2 = (0 0 0 1)
vector3 = (1 0 0 0)
vector4 = (0 0 1 1)

The maximum no. of clusters to be formed is m=2 with learning rate $\propto = 0.6$

Sol:

With only 2 clusters available, the neighborhood of nodJ is set so that only one cluster up dates its weight at each step

Initial weight matrix:

$$\begin{bmatrix} 0.2 & 0.8 \\ 0.6 & 0.4 \\ 0.5 & 0.7 \\ 0.9 & 0.3 \end{bmatrix}$$

1- for the first vector

x_1	x_2	x_3	x_4
(1	1	0	0)

$$D(i) = (1-0.2)^2 + (1-0.6)^2 + (0-0.5)^2 + (0-0.9)^2 = 1.86$$

$$D(2) = (1-0.8)^2 + (1-0.4)^2 + (0-0.7)^2 + (0-0.3)^2 = 0.98 \text{ (Minimum)}$$

$\therefore J = 2$ (The input vector) is closest to output node 2)

$\therefore$ The weight on the winning unit is update:-

$$W_{21}(\text{new}) = W_{12}(\text{old}) + 0.6(x_i - W_{12}(\text{old}))$$
$$= 0.8 + 0.6(1 - 0.8) = 0.92$$

$$W_{22}(new) = 0.4 + 0.6(1 - 0.4)$$
$$= 0.4 + 0.36 = 0.76$$

$W_{23}(\text{new}) = 0.7 + 0.6(0-0.7)$

$= 0.28$

$W_{24}(\text{new}) = 0.3 + 0.6(0-0.3)$

$= 0.12$

This gives the weight matrix $\begin{bmatrix} 0.2 & 0.92 \\ 0.6 & 0.76 \\ 0.5 & 0.28 \\ 0.9 & 0.12 \end{bmatrix}$

2-for the second vector $(0 \quad 0 \quad 0 \quad 1)$

$D(i) = (0-0.2)^2 + (0-0.6)^2 + (0-0.5)^2 + (1-0.9)^2 = 0.66$ minimum

$D(2) = (0-0.92)^2 + (0-0.76)^2 + (0-0.28)^2 + (1-0.12)^2 = 2.2768$

$\therefore J = 1$(The i/p vector is closest to o/p node 1)

After update the first column of the weight matrix:-

$$\begin{bmatrix} 0.08 & 0.92 \\ 0.24 & 0.76 \\ 0.20 & 0.28 \\ 0.96 & 0.12 \end{bmatrix}$$

3- for the third vector (1 0 0 0)

$D(i) = (-0.08)^2 + (0-0.24)^2 + (0-0.20)^2 + (0-0.96)^2 = 1.856$

$D(2) = (1-0.92)^2 + (0-0.76)^2 + (0-0.28)^2 + (1-0.12)^2$

$= 2.2768$ minimum

$\therefore J = 2$ (The i/p vector is closest to o/p node (2))

After update the second column of the weight matrix:-

$$\begin{bmatrix} 0.08 & 0.968 \\ 0.24 & 0.304 \\ 0.20 & 0.112 \\ 0.96 & 0.0.4_8 \end{bmatrix}$$

4- for the fourth vector (0 0 1 1)

$D(i) = (0 - 0.08)^2 + (0 - 0.24)^2 + (1 - 0.20)^2 + (1 - 0.96)^2 = 0.7056$ minimum

$D(2) = (0 - 0.968)^2 + (0 - 0.304)^2 + (1 - 0.112)^2 + (1 - 0.048)^2 = 2.724$

$\therefore J = 1$(the i/p vector is closest to o/p node 1)

After update the first column of the weight matrix :-

$$\begin{bmatrix} 0.032 & 0.968 \\ 0.096 & 0.304 \\ 0.680 & 0.112 \\ 0.984 & 0.048 \end{bmatrix}$$

$\therefore$ Reduce the learning rate

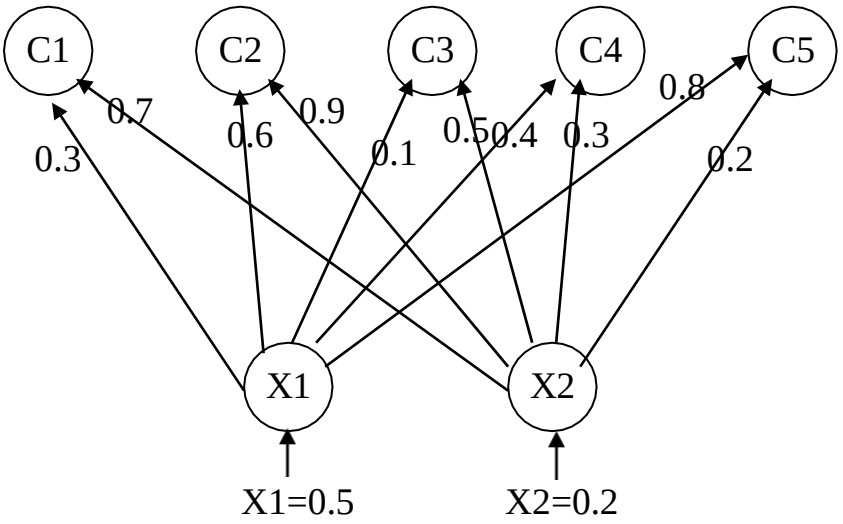
$\propto (t+1) * \propto (t) = 0.5 * (0.6) = 0.3$

$\therefore$ After one iteration the weight matrix will be:-

$$\begin{bmatrix} 0.032 & 0.970 \\ 0.096 & 0.300 \\ 0.680 & 0.110 \\ 0.984 & 0.048 \end{bmatrix}$$

H.W

Find the output node with minimum distance then update its reference vector only $\propto = 0.5$



3.6 Self- Organizing Networks

Self –organizing networks mean that the systems are trained by showing examples of patterns that are to be classified, and the network is allowed to produce its own output code for the classification.

In self – organizing networks the training can be supervised or unsupervised. The advantage of unsupervised learning is that the network finds its own energy minima and therefore tends to be more efficient in terms of the number of patterns that it can accurately store and recall.

In self – organizing networks four properties are required:-

1- The weight in the neurons should be representative of a class of patterns. So each neuron represents a different class
2- Input patterns are presented to all of the neurons, and each neuron produces an output. The value of the output of each neuron is used as a measure of the match between the input pattern and the pattern stored in the neuron
3- A competitive learning strategy which selects the neuron with the largest response.
4- A method of reinforcing the largest response.

3.7 Adaptive Resonance Theory (ART)

Adaptive resonance theory (ART) was developed by Carpenter and Grossberg (1987). One form, ART 1, is designed for clustering binary vectors, another, ART2 also by Carpenter and Grossberg (1987).

These nets cluster inputs by using unsupervised learning input patterns may be presented in any order. Each time a pattern is presented, an appropriate cluster unit is chosen and that cluster's weights are adjusted to let the cluster unit learn the pattern.

3.7.1 Basic Architecture

Adaptive resonance theory nets are designed to allow the user to control the degree of similarity of patterns placed on the same cluster. ART1 is designed to cluster binary input vectors. The architecture of an ART1 net Consists of the following units:-

1- Computational units.

2- Supplemental units.

1- Computational units:-

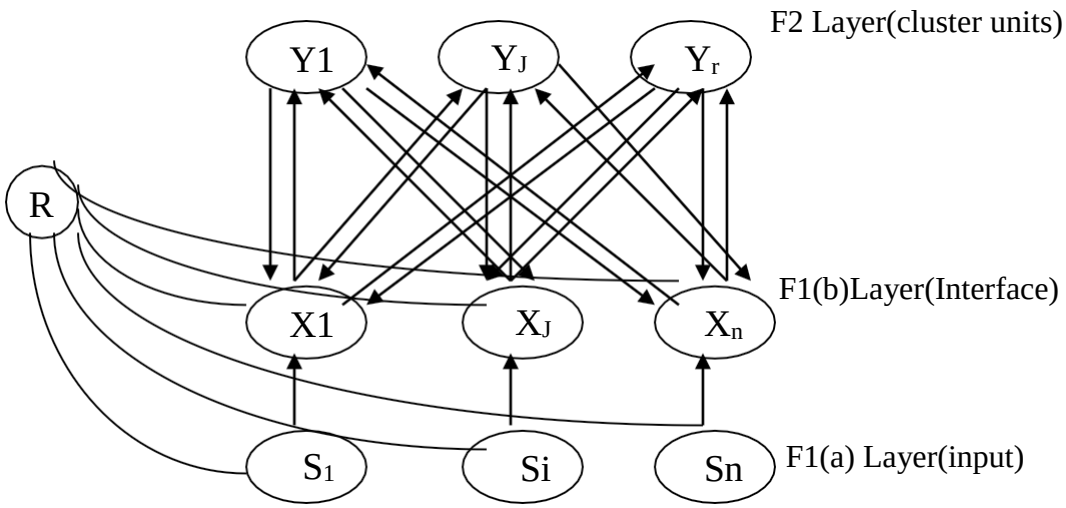
The architecture of the computational units for ART1 consists of three field of unites:-

1- The F1 units (input and interface units)

2- The F2 units (cluster units)

3- Reset unite

This main portion of the ART1 architecture is illustrated in figure bellow:-



"Basic structure of ART1"

The F1 layer can be considered to consist to two of two parts:-

1- F1 (a) the input units

2- F1 (b) the interface units.

Each unit in the F1 (a) (input) layer is connected to the corresponding unit in the F1 (b) (interface) layer .Each unit in the F1 (a) &F1 (b) layer is connected to the reset unit, which in turn is connected to every F2 unit. Each unit in the F1 (b) is connected to each unit in the F2 (cluster) by two weighted pathways:-

1- Bottom –up weights:-

The F1(b) unit Xi is connected to the F2 unit Yj by bottom –up weights bij.

2- Top- down weights:-

Unit Yj is connected to unit Xi by top-down weights tji.

The F2 layer is a competitive layer in which only the uninhibited node with the largest net input has a non zero activation.

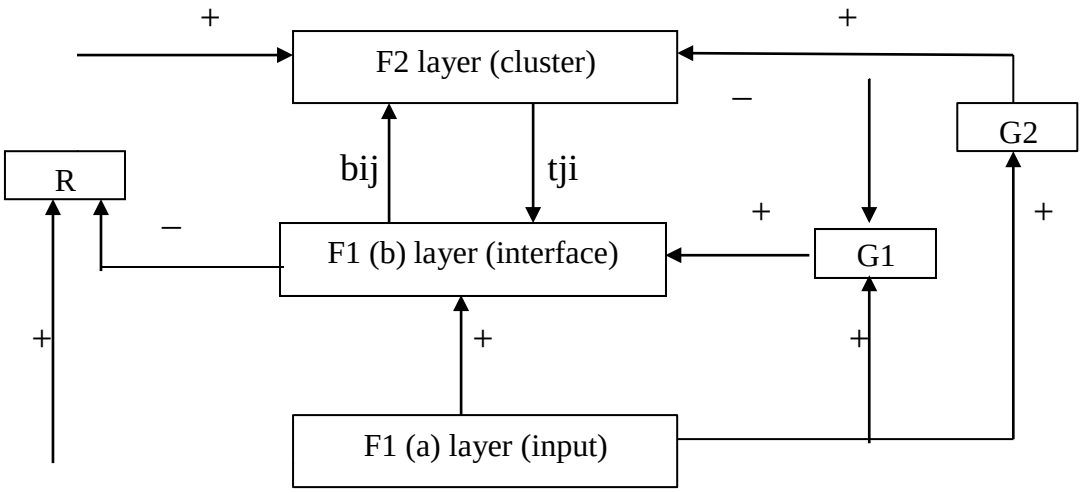
2-Supplemental Units:-

The Supplemental Units shown in figure (4-6) are important from a theoretical point of view. There are two Supplemental Units called gain control units, these are:-

1- Gain1 ⟶ g1 or G1

2- Gain2 ⟶ G2

In addition to the reset unit R



"The Supplemental Units for ART1"

Excitatory signals are indicated by (+) and inhibitory signals by (-), a signal is sent whenever any unit in the designated layer is (on).

Each unit in either the F1 (b) or F2 layer of the ART1 net has three sources from which it can receive a signal

1- F1(b) can receive signals from :-
   - F1(a) (an input signal)
   - F2 node (top –down signal)
   - G1 unit.
2- F2 unit can receive a signal from :-
   - F1 (b) (an interface unit)
   - R unit (reset unit)
   - G2 unit

An F1(b) (interface) or F2 unit must receive two excitatory signals in order to be (on). Since there are three possible sources of signals, this requirement is called the *two- thirds rule*.

The reset unit R controls the *vigilance* matching (the degree of similarity required for patterns to be assigned to the same cluster unit is controlled by a user – specified parameter, known as the vigilance parameter). When any unit in the F1 (a) is on, an excitatory signal is sent to R. the strength of that signal depends on how many F1(a) are (on). R also receives inhibitory signals from the F1(b) that are (on). If enough F1(b) are (on), unit R is prevented from firing . If unit R does fire, it inhibits any F2 unit that is (on). This forces the F2 layer to choose a new winning node.

There are two types of learning that differ both in their theoretical assumptions and in their performance characteristics can be used for ART nets:-

fast learning

It is assumed that weight updates during resonance occur rapidly, in fast learning, the weight reach equilibrium on each trial. It is assumed that the ART1net is being operated in the fast learning mode.

slow learning

The weight changes occur slowly relative to the duration of a learning trial, the weights do not reach equilibrium on a particular trail.

H.W

Q1: Write the complete algorithm for kohonen neural network?

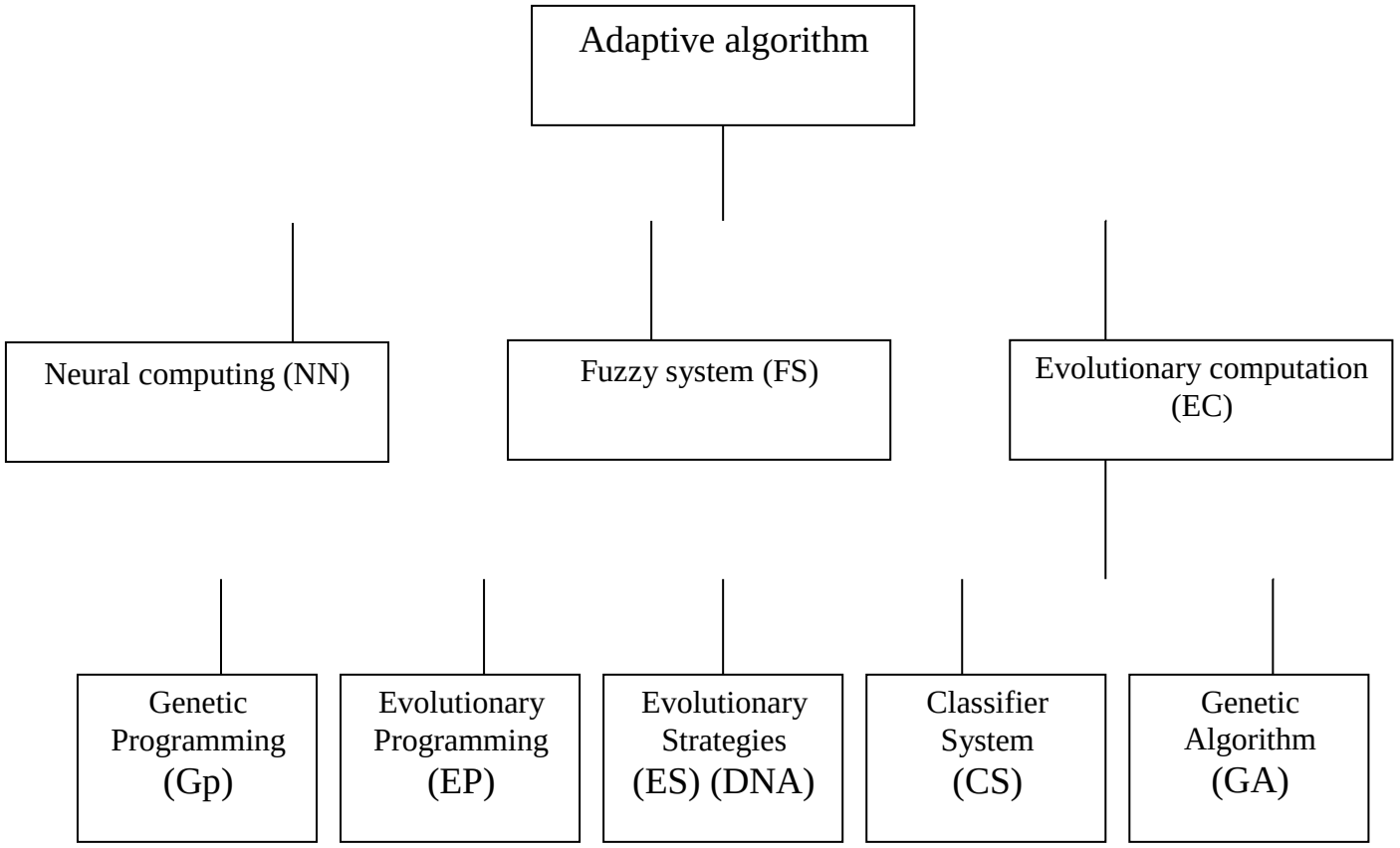
Q2: there are two basic units in ART1 architecture, list them and draw the figure for each one of them.

Q3: there are two kinds of learning in ART neural network. Briefly explain each one of them. Which kind does ART1 use?

Q4: Define the following expressions:-

1- Euclidean Distances
2- Vigilance matching
3- Bottom –up and top- down weights
4- Two-thirds rule

4.1 Genetic Algorithms (GA)



Structure of Adaptive Algorithm

A genetic algorithm is a search procedure modelled on the mechanics of natural selection rather than a simulated reasoning process. Domain Knowledge is embedded in the abstract representation of a candidate solution termed an organism. Organisms are grouped into sets called populations. Successive population are called generation. The aim of GA is search for goal.

A generational GA creates an initial generation G(0) , and for each generation ,G(t) , generates a new one ,G(t+1) . An abstract view of the algorithm is:-

Generate initial population, G(0);

Evaluate G(0);

t:=0;

Repeat

t:=t+ 1

Generate G(t) using G(t-1);

Evaluate G(t);

Until solution is found.

4.1.1 Genetic Operators

The process of evolving a solution to a problem involves a number of operations that are loosely modeled on their counterparts from genetics .

Modeled after the processes of biological genetics , pairs of vectors in the population are allowed to " mate" with a probability that is proportional to their fitness . the mating procedure typically involves one or more genetic operators . The most commonly applied genetic operators are :-

1- Crossover.
2- Mutation.
3- Reproduction.

1- Crossover

Is the process where information from two parents is combined to form children. It takes two chromosomes and swaps all genes residing after a randomly selected crossover point to produce new chromosomes.

This operator does not add new genetic information to the population chromosomes but manipulates the genetic information already present in the mating pool (MP).

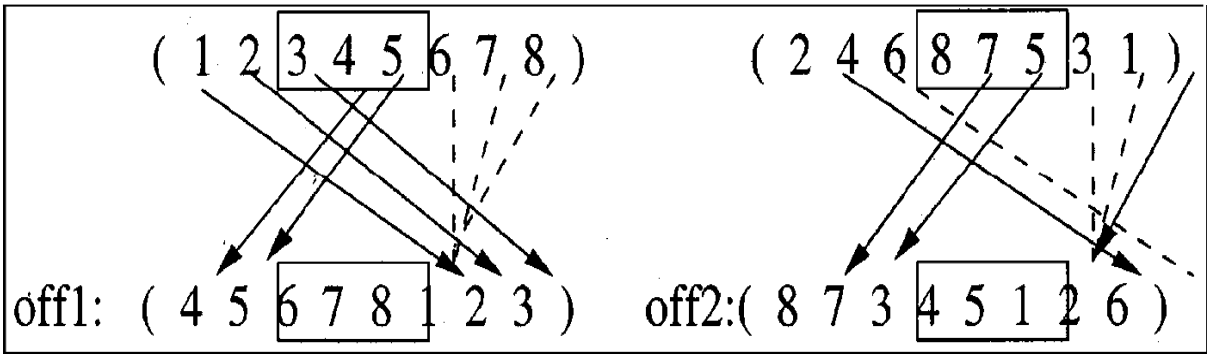
The hope is to obtain new more fit children It works as follows :-

1- Select two parents from the MP (The best two chromosomes) .
2- Find a position K between two genes randomly in the range (1, M-1)
   M = length of chromosome
3- Swap the genes after K between the two parents.

The output will be the both children or the more fit one.

1- a Order crossover (OX1)

The *order crossover operator* (Figure 4) was proposed by Davis (1985). The OX1 exploits a property of the path representation, that the order of cities (not their positions) are important. It constructs an offspring by choosing a sub tour of one parent and preserving the relative order of cities of the other parent.



Order crossover (OX1)

For example, consider the following two parent tours:

(1 2 3 4 5 6 7 8) and
(2 4 6 8 7 5 3 1),

and suppose that we select a first cut point between the second and the third bit and a second one between the fifth and the sixth bit. Hence,

(1 2|3 4 5|6 7 8) and
(2 4|6 8 7|5 3 1).

The offspring are created in the following way. First, the tour segments between the cut point are copied into the offspring, which gives

$$(* *|3\ 4\ 5|* * *) \text{ and}$$
$$(* *|6\ 8\ 7|* * *).$$

Next, starting from the second cut point of one parent, the rest of the cities are copied in the order in which they appear in the other parent, also starting from the second cut point and omitting the cities that are already present. When the end of the parent string is reached, we continue from its first position. In our example this gives the following children:

$$(8\ 7|3\ 4\ 5|1\ 2\ 6) \text{ and}$$
$$(4\ 5|6\ 8\ 7|1\ 2\ 3),$$

- *Partially mapped crossover (PMX)*
- *Cycle crossover (CX)*
- *Order based crossover (OX2)*
- *Position based crossover (POS)*
- *Heuristic crossover*
- *Genetic edge recombination crossover (ER)*
- *Sorted match crossover*
- *Maximal preservative crossover (MPX)*
- *Voting recombination crossover (VR)*
- *Alternating position Crossover (AP)*

2- Mutation

The basic idea of it is to add new genetic information to chromosomes. It is important when the chromosomes are similar and the GA may be getting stuck in Local maxima. A way to introduce new information is by changing the allele of some genes. Mutation can be applied to :-

1- Chromosomes selected from the MP.

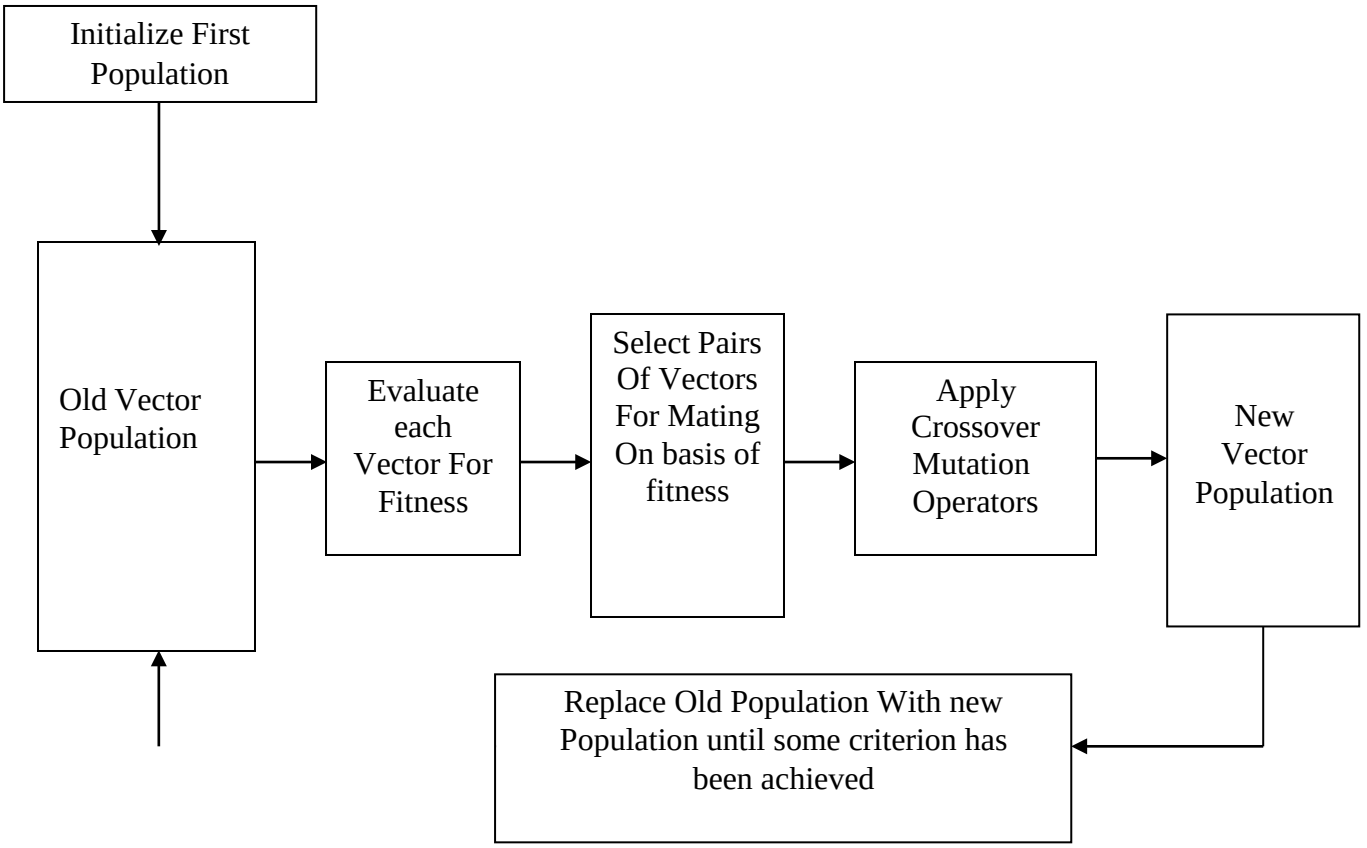
2- Chromosomes that have already subject to crossover.

The Figure bellow illustrates schematically the GA approach.

3- Reproduction

After manipulating the genetic information already present in the MP by fitness function the reproduction operator add new genetic information to the population of the chromosomes by combining strong parents with strong children , the hope is to obtain new more fit children . Reproduction imitate to the natural selection.

This schematic diagram of a genetic algorithm shows the functions that are carried out in each generation. Over a number of such generation the initial population is evolved to the point where it can meet some criterion with respect the problem at hand .



"Genetic Algorithm approach "

4.2 Genetic Programming (GP)

Genetic programming (GP) is a domain – independent problem – solving approach in which computer programs are evolved to solve, or approximately solve problems. Thus, it addresses one of the central goals of computer science namely automatic programming. The goal of automatic programming is to create, in an automated way, a computer program that enables a computer to solve a problem.

GP is based on reproduction and survival of the fittest genetic operations such as crossover and mutation. Genetic operation are used to create new offspring population of individual computer programs from the current population of programs .

GP has several properties that make it more suitable than other paradigms (e.g. . best – first search , heuristic search , hill climbing etc .) , these properties are :-

1- GP produces a solution to a problem as a computer program. Thus GP is automatic programming.
2- Adaptation in GP is general hierarchical computer programs of dynamically varying size & shape.
3- It is probabilistic algorithm.
4- Another important feature of GP is role of pre processing of inputs and post processing of outputs .

To summarize, genetic programming includes six components, many very similar to the requirements fo GAs:

1- A set of structures that undergo transformation by genetic operators.
2- A set of initial structures suited to a problem domain.
3- A fitness measure, again domain dependent, to evaluate structures.
4- A set of genetic operators to transform structures.

5- Parameters and state descriptions that describe members of each generation.

6- A set of termination conditions.

EX. 11:-

By using GA step by step, find the maximum number in 0 to 31.let k=3 and population size=4 ,and the initial population is:-

14 ⟶ 01110
3 ⟶ 00011
25 ⟶ 11001
21 ⟶ 10101

} population

Fitness function will be:-

25&21

3&14

25&21

1 1 | 0 0 1 ⟶ 25
1 0 | 1 0 1 ⟶ 21

1 1 1 0 1 ⟶ 29
1 0 0 0 1 ⟶ 17

14&3

0 1 | 1 1 0 ⟶ 14
0 0 | 0 1 1 ⟶ 3

0 1 0 1 1 ⟶ 11
0 0 1 1 0 ⟶ 6

the new population will be an array and we choose position [16] randomly to do mutation on it:-

1	1	1	0	1
1	0	0	0	1
0	1	0	1	1
0	0	1	1	0

Mutation ⟶

1	1	1	0	1
1	0	0	0	1
0	1	0	1	1
1	0	1	1	0

Mutation 0 ⟶ 1

:

After Mutation the new population will be:

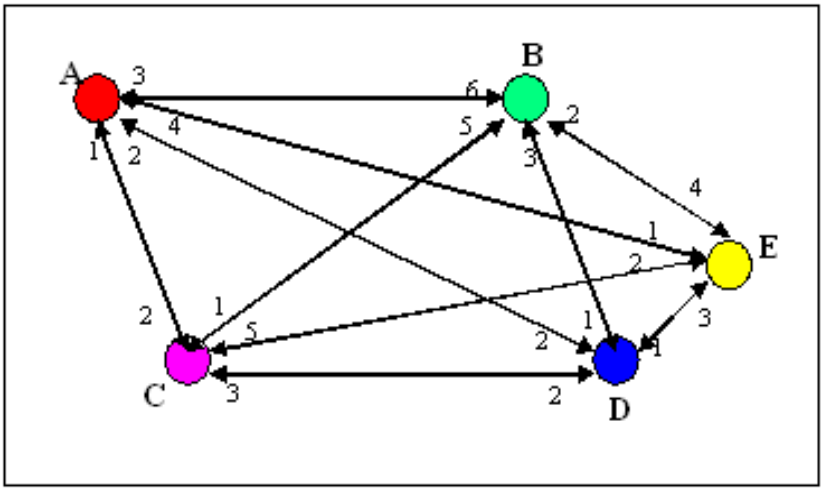
1 1 1 0 1 = 29
1 1 0 0 1 =25
1 0 1 0 1 =21
1 0 1 1 0 =22

} reproduction

Because the mutation we replace 17 with 22 in the new population.

EX 12

Apply GA in travelling salesman to find the shortest path . let k=2 and the initial population is:-



A B C D E =12
B C D E A =10
A C D B E =11
E C A D B =11
B A D C E =10
D E B A C =10

} initial population

B C D | E A =10
B A D | C E =10

B C D A E = 6
B A D E C =13

D E B | A C =10
A C D | B E

D E B A C = 10
A C D E B =9

E C A | D B =11
A B C | D E =12

E C A D B = 11
A B C D E =12

THE NEW POPULATION IS:-

B C D A E = 6
B A D E C =13
D E B A C = 10
A C D E B = 9
E C A D B =11
A B C D E =12

أما زأخه افضل الكروموسومات من الجيل الجديد و انضل الكروموسومات من الجيل السابق)بنفس عمد المجتمع (ونعملcrossover لهما او نعمل crossoverالجيل الجديد نفط

H.W

Q1: Can the bit string 0 1 0 1 0 1 0 1 be the result of crossing over the following pairs of parents?:-

a- 11111111 and 00000000

b-01010101 and 11111111

c-10100101 and 01011010

Q2: What is genetic algorithm (GA). Explain its algorithm.

Q3: What are the most commonly operators used in GA, list it only, then draw the figure which illustrates schematically the GA approach.

Q4: Adaptive algorithm includes GA and GP in one port of it. Illustrates schematically the main structure of adaptive algorithm.

Resources

www.uotechnology.edu.iq/dep-cs