Complexity in social dynamics : from the micro to the macro
Laboratory 2 – Neural networks

Franco Bagnoli

Namur – 11/4/2008

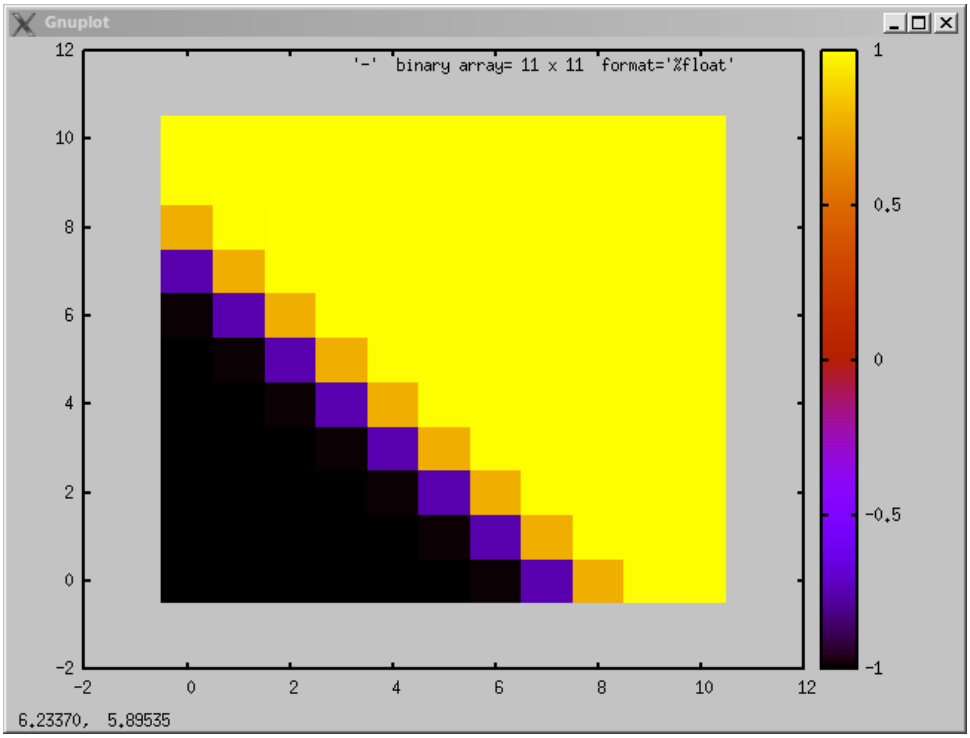
1 Laboratory 2

Neural networks

1. Perceptron. Linear classification.
2. Perceptron learning.
3. Multi-layer perceptron.
4. Backpropagation
5. Unsupervised learning (to do).
6. Hopfield model (to do).

Perceptron

- The simple perceptron with two inputs is characterized by the two weights w_1 and w_2, and the threshold b (`perceptron.f90`)
- Experiment with the program and discover the role of parameters.



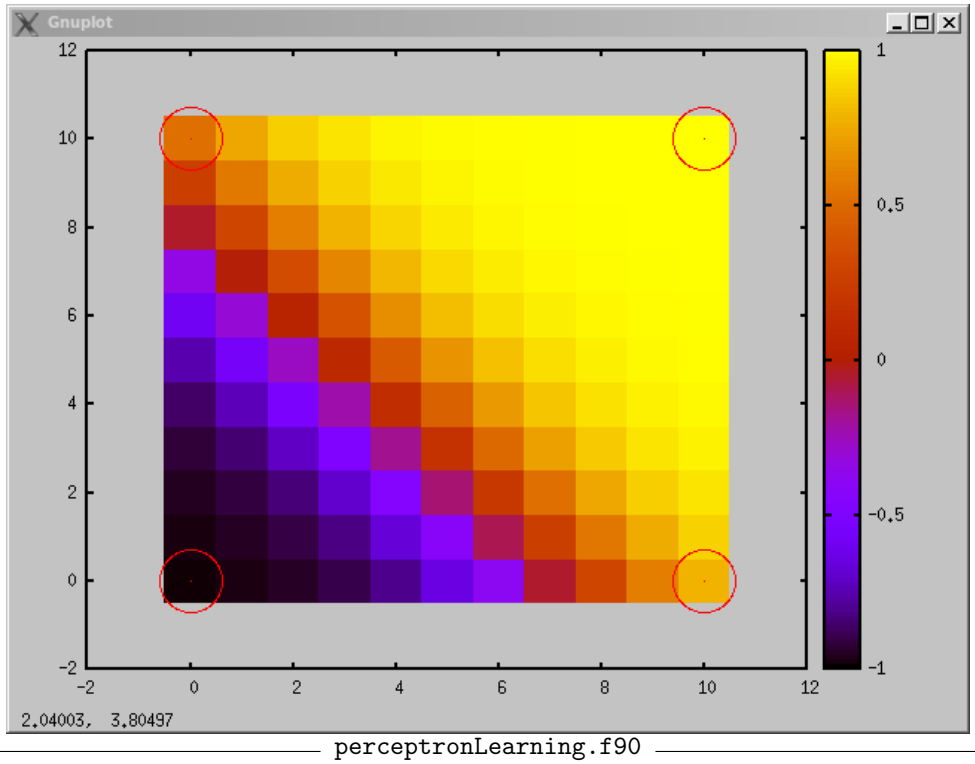
perceptron.f90

```
program perceptron
  implicit none
  integer, parameter :: N=10
  real :: p(0:N,0:N)
  integer :: i, j
  real :: x1, x2, w1, w2, y, b, T
  character*200 :: str

  call gnuplotOpen("gnuplot")
  call gnuplotExecute("set mouse; set term x11")
  call gnuplotExecute("set cbrange [-1:1]")
```

```
  T = .1

  do while (.true.)
    print *, "give me w1, w2, b"
    read (*,*) w1, w2, b
    print *, "using:", w1, w2, b
    do i=0,N
      do j=0, N
       x1 = 2*real(i)/N - 1
       x2 = 2*real(j)/N - 1
       y = tanh((x1*w1+x2*w2+b)/T)
       p(j,i) = y
      end do
    end do
    write (str, *) "plot '-' "// &
    " binary array=",N+1,"x",N+1," format='%float'", &
    " with image ", char(0)
    call gnuplotExecute(str)
    call gnuplotWrite(P, sizeof(p))
    call gnuplotFlush()
  end do
  call gnuplotClose()
end program
```

Perceptron learning

- Perceptron can "learn" how to discriminate about linearly separable examples (`perceptronLearning.f90`)
- However, it is not able to learn how to separate the inputs into disconnected sets (the "XOR" problem).



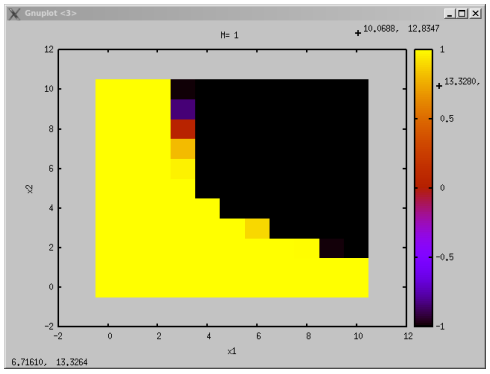
perceptronLearning.f90

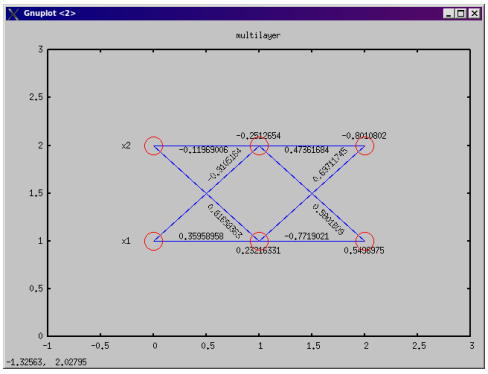
```
program perceptronLearning
  implicit none
```

```
  integer, parameter :: N=10
  real :: p(0:N,0:N)
  integer :: i, j, k, kk
  real :: x1, x2, w1, w2, y, b, T
  integer, parameter :: NE=20 !number of examples
  real :: xx1(NE), xx2(NE), yy(NE)
  character*200 :: str
  real :: eta=.1


  call gnuplotOpen("gnuplot")
  call gnuplotExecute("set mouse; set term x11; unset key")
  call gnuplotExecute("set cbrange [-1:1]")

  T = .5

  call random_number(w1)
  call random_number(w2)
  call random_number(b)
  do k=1, NE
    print *, "give me an example: x1, x2, class"
    read (*,*) xx1(k), xx2(k), yy(k)
    y = tanh((xx1(k)*w1+xx2(k)*w2+b)/T)
    print *, "required: ", yy(k), " computed: ", y
    ! delta for continous learning
!    w1 = w1 + eta*(1-y**2)*(yy(k)-y)*xx1(k)/T
!    w2 = w2 + eta*(1-y**2)*(yy(k)-y)*xx2(k)/T
!    b = b + eta*(1-y**2)*(yy(k)-y)/T
    ! delta for binary learning
    w1 = w1 + eta*(yy(k)-y)*xx1(k)
    w2 = w2 + eta*(yy(k)-y)*xx2(k)
    b =   b + eta*(yy(k)-y)
    print *, "parameters now are: ", w1, w2, b
    do i=0,N
      do j=0, N
       x1 = 2*real(i)/N - 1
       x2 = 2*real(j)/N - 1
       y = tanh((x1*w1+x2*w2+b)/T)
       p(i,j) = y
      end do
    end do
    write (str, *) "plot '-' "// &
    " binary array=",N+1,"x",N+1," format='%float'", &
    " with image , '-' w p pt 6 ps 10", char(0)
    call gnuplotExecute(str)
    call gnuplotWrite(P, sizeof(p))
    do kk=1,k
      write(str, *) (xx1(kk) +1)*N/2, (xx2(kk)+1)*N/2, char(0)
      call gnuplotExecute(str)
    end do
    call gnuplotExecute("end")
    call gnuplotFlush()
  end do
  call gnuplotClose()
end program
```

Multi-layer perceptron

- By adding more layer one can classify arbitrary regions (`multiPerceptron.f90`)
- The problem is how to design the weights and the thresholds.
- Notice that when the number of layer is large, the output is homogeneous. This is because the random weights and threshold adds to give about zero, so the last b dominates.





multiPerceptron.f90

```
program multiPerceptron
  implicit none
  integer, parameter :: N = 10 ! resolution is input space
  integer :: M ! number of hidden layers
  real :: p(0:N,0:N)
  integer :: i, j, k, q
  real, allocatable :: x(:,:), w(:,:,:), b(:,:)! layers, inputs = x(0)
                                               ! output = x(M+1,1)
  real :: T
  character*200 :: str

  call gnuplotOpen("gnuplot")
  call gnuplotExecute("set mouse; set term x11")
  call gnuplotExecute("set cbrange [-1:1]")
  call gnuplotExecute("set xlabel 'x1'; set ylabel 'x2'; unset key")

  ! for showing the net
  call gnuplotSelect(1)
  call gnuplotOpen("gnuplot")
  call gnuplotExecute("set term x11; set nokey")
  call gnuplotExecute("set title 'multilayer'")
  call gnuplotSelect(0)


  call random_seed()
  T = .1


  do while (.true.)
    print *, "number of hidden layers"
    read (*,*) M
    print *, "using ", M, "  hidden layers"
    allocate(x(0:M+1,2))
    allocate(w(M+1,2,2))
    allocate(b(M+1,2))
    ! random initialization
```

```
call random_number(w)
w = 2*w-1 ! between -1,1
call random_number(b)
b = 2*b-1


! displaying the mapping

do i=0,N
  do j=0, N
    x(0,1) = 2*real(i)/N - 1
    x(0,2) = 2*real(j)/N - 1
    do k=1, M+1  ! layer
      do q=1,2 ! neurons in a layer
        x(k,q) = tanh(( &
          x(k-1,1)*w(k,q,1)+x(k-1,2)*w(k,q,2)+b(k,q) &
          )/T)
      end do
    end do
    p(i,j) = x(M+1,1)
  end do
end do

write(str,*) "set title 'M=",M,"'"
call gnuplotExecute(str)
write (str, *) "plot '-' "// &
" binary array=",N+1,"x",N+1," format='%float'", &
" with image ", char(0)
call gnuplotExecute(str)
call gnuplotWrite(P, sizeof(p))
call gnuplotFlush()

! displaying the network

call gnuplotSelect(1)
call gnuplotExecute("unset label; unset arrow")
write (str, *) "set yrange [0:3]"
call gnuplotExecute(str)
write (str, *) "set xrange [-1:",M+2,"]"
call gnuplotExecute(str)
call gnuplotExecute("set label 'x1' at -.3,1")
call gnuplotExecute("set label 'x2' at -.3,2")
write(str,*) "set label 'y' at ",M+1+0.3,",1", char(0)
do i=1, M+1
  do k=1,2
    do q=1,2
      write (str, *) "set arrow from ",i-1,",",k," to ",i,",",q, &
        " nohead linestyle 3 " , char(0)
      call gnuplotExecute(str)
      write (str, *) "set label '",w(i,k,q), &
        "' at ",i+abs(q-k)*0.3-0.8,",",k+(q-k)*0.4 - &
        (1-abs(q-k))*(k-1.5)*0.1, &
         "rotate by ",(k-q)*45, " ", char(0)
      call gnuplotExecute(str)
    end do
    call gnuplotExecute(str)
```

```
        write(str, *) "set label '",b(i,k),"' at ",i,",",&
            k+(k-1.5)*0.2 , " center ", char(0)
        call gnuplotExecute(str)
      end do
    end do
    call gnuplotExecute("plot '-' with points ps 6 pt 6")
    do i=0, M+1
      do k=1,2
        write(str, *) i, k, char(0)
        call gnuplotExecute(str)
      end do
    end do
    call gnuplotExecute("end")
    call gnuplotFlush()
    call gnuplotSelect(0)


    deallocate(x)
    deallocate(w)
    deallocate(b)
  end do
  call gnuplotClose()
end program
```

Supervised learning

- Supervised learning means changing weights and threshold in order to minimise the error (difference between the expected output $\overline{y}^{(k)}$ and the actual one $y^{(k)}(x_1^{(k)}, x_2^{(k)})$).
- It is a minimization problem similar to least squares. Since however the "perceptron" function is in general not invertible, one has to use iterative techniques.
- One of these techniques is gradient descent. Compute the gradient of the function to be minimized and take a step in the opposite direction (backpropagation). Another possibility is simulated annealing.
- In gradient descent there are two possible problems: the choice of the step and the "insensibility" of weights that have value near 1 or -1.

Backpropagation 1

The increase in error $\Delta^{(k)}\xi^2 = \left(\overline{y}^{(k)} - y^{(k)}\right)^2$ due to example k is (neglecting k)

$$\frac{\partial \Delta\xi^2}{\partial w} = 2(\overline{y} - y)\frac{\partial y}{\partial w},$$

where

$$x_i(\ell) = \tanh\left(\sum_j w_{ij}x_j(\ell - 1) + b_i(\ell)\right),$$

and $x_i = x_i(0)$, $y = x_1(L + 1)$ and $\ell \in \{1, \ldots, L\}$ numbers the layers.

Backpropagation 2

The gradient descent gives for the w a recursive formula. First compute the errors backwards

$$\delta_j(\ell) = (1 - x_j^2(\ell))\sum_i w_{ij}(\ell + 1)\delta_i(\ell + 1),$$

(remember that the derivative of $\tanh(x)$ is $1 - tanh^2(x)$), and then

$$w'_{ij}(\ell) = w_{ij}(\ell) + \eta x_j(\ell - 1)\delta_i(\ell).$$

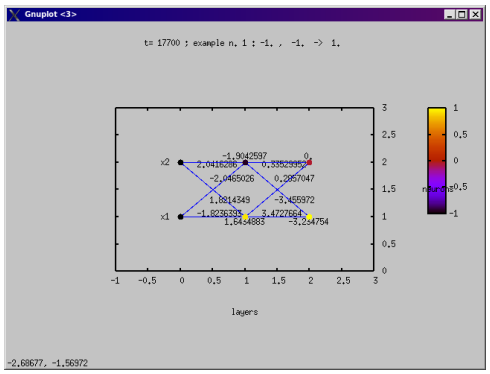
For the weights

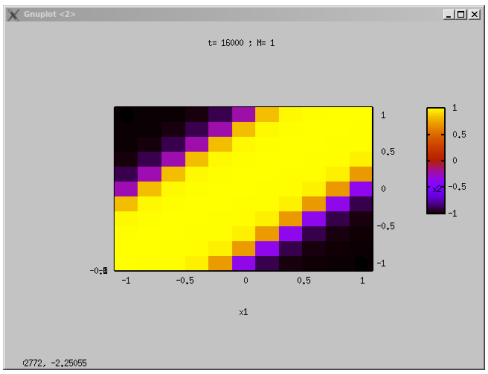
$$b'_i(\ell) = b_i(\ell) + \eta\delta_i(\ell).$$

The parameter η is the learning speed, and determines the width of a step in the direction of gradient descent.

Backpropagation implementation

- Since there are many inputs (the examples) we read them from a file (`multiPerceptronLearning.f90`, `xor.inp`, `and.inp`, etc.)
- One can check that with just an hidden layer, one can learn the xor rule.
- However, if one starts with a network with weights -1 or 1, the convergence is quite slow (insensitivity).
- Sometimes, backpropagation fails...





multiPerceptronLearning.f90

```
program multiPerceptron
  implicit none
  integer :: N       ! resolution in input space
  integer :: M            ! number of layers
  real,allocatable :: p(:,:) ! input phase space
  integer :: i, j, k, q, ke, kk, ii
  real,allocatable :: x(:,:), w(:,:,:), b(:,:)! layers, inputs = x(0)
                                              ! output = x(M,1)
  real :: T
  character*200 :: str
  integer :: NE ! number of examples
  real, allocatable :: xx(:,:), yy(:)
  real,allocatable :: delta(:,:)
  real :: eta
  integer :: NN ! repetitions
  real ::error

  if (iargc() < 1) then
    print *, "parameter file?"
    read (*,*) str
  else
    call getarg(1,str)
  end if
  open (unit=12, file=str)
```

```
read (12,*), T
print *, "temperature= ", T
read (12,*), eta
print *, "eta= ", eta
read (12,*), N
print *, "resolution in phase space= ", N
read (12,*), M
print *, "number of hidden layers= ", M
read (12,*), NE
print *, "number of examples= ", NE
read (12,*), NN
print *, "example repetitions= ", NN

allocate(x(0:M+1,2))
allocate(w(M+1,2,2))
allocate(b(M+1,2))
allocate(delta(M+1,2))
allocate(xx(NE, 2))
allocate(yy(NE))
allocate(p(0:N,0:N))


do ke=1,NE
  read (12,*) xx(ke,1), xx(ke,2), yy(ke)
  print *, "example n. ",ke,"=",xx(ke,1), xx(ke,2), yy(ke)
end do
close(12)

call gnuplotOpen("gnuplot")
call gnuplotExecute("set mouse; set term x11; unset key")
call gnuplotExecute("set view 0,0,1; set cbrange [-1:1]")
call gnuplotExecute("set xrange [-1.1:1.1]")
call gnuplotExecute("set yrange [-1.1:1.1]")
call gnuplotExecute("set xlabel 'x1'")
call gnuplotExecute("set ylabel 'x2'")

! for showing the net
call gnuplotSelect(1)
call gnuplotOpen("gnuplot")
call gnuplotExecute("set mouse; set term x11; unset key")
call gnuplotExecute("set xlabel 'layers'; set ylabel 'neurons'")
call gnuplotExecute("set title 'back-propagation'; unset ztics")
call gnuplotExecute("set view 0,0,1; set cbrange [-1:1]")
call gnuplotSelect(0)

call random_seed()

call random_number(w)
w = 0.6*(2*w-1)
call random_number(b)
b = 0


do ii=1, NN
  error=0
  do ke=1,NE
```

```
!computing the output
x(0,:) = xx(ke,:)
do k=1, M+1  ! layer
  do q=1,2 ! row
    x(k,q) = tanh(( &
      x(k-1,1)*w(k,q,1)+x(k-1,2)*w(k,q,2)+b(k,q) &
      )/T)
  end do
end do
!print *, "required: ", yy(ke), " computed: ", x(M+1,1)
error = error + (yy(ke)-x(M+1,1))**2

! back-propagating the error
! see
!http://galaxy.agh.edu.pl/~vlsi/AI/backp_t_en/backprop.html
! but they are wrong in computing delta,
!see neuron-K7 pag. 18

delta(M+1,1) =  (1-x(M+1,1)**2)*(yy(ke)-x(M+1,1))
delta(M+1,2) =  0 ! this is fictitious
do k=M,1 ! layer
  do q=1,2
    delta(k,q) = (1-x(k,q)**2)*(w(k+1,1,q)*delta(k+1,1) + &
      w(k+1,2,q)*delta(k+1,2))
  end do
end do

! and changing weights

do k=1,M+1 ! layer
  do q=1,2
    w(k,q,1) = w(k,q,1) + eta*x(k-1,1)*delta(k,q)
    w(k,q,2) = w(k,q,2) + eta*x(k-1,2)*delta(k,q)
    b(k,q) = b(k,q) + eta*delta(k,q)
  end do
end do
if (mod(ii,100) == 0) then
  ! show network
  call gnuplotSelect(1)
  write(str,*) "set title 't=",ii,"; example n.",&
    ke,":",xx(ke,1),", ",xx(ke,2)," -> ",yy(ke),"'"
  call gnuplotExecute(str)
  call gnuplotExecute("unset label; unset arrow")
  write (str, *) "set yrange [0:3]"
  call gnuplotExecute(str)
  write (str, *) "set xrange [-1:",M+2,"]"
  call gnuplotExecute(str)
  call gnuplotExecute("set label 'x1' at -.3,1")
  call gnuplotExecute("set label 'x2' at -.3,2")
  write(str,*) "set label 'y' at ",M+1+0.3,",1", char(0)
  do i=1, M+1
    do k=1,2
      do q=1,2
        write (str, *) "set arrow from ",i-1,",",k," to ",i,",",q, &
          " nohead linestyle 3 " , char(0)
        call gnuplotExecute(str)
```

```
            write (str, *) "set label '",w(i,k,q), &
              "' at ",i+abs(q-k)*0.2-0.8,",",k+(q-k)*0.3 - &
              (1-abs(q-k))*(k-1.5)*0.1,  char(0)
            call gnuplotExecute(str)
          end do
          write(str, *) "set label '",b(i,k),"' at ",i,",",&
              k+(k-1.5)*0.2 , " center ", char(0)
          call gnuplotExecute(str)
        end do
      end do
      call gnuplotExecute("splot '-'  with points ps 2 pt 7 lc palette")
      do i=0, M+1
        do k=1,2
          write(str, *) i, k, x(i,k), char(0)
          call gnuplotExecute(str)
        end do
      end do
      call gnuplotExecute("end")
      call gnuplotFlush()
      call gnuplotSelect(0)

      ! show learning

      do i=0,N
        do j=0, N
          x(0,1) = 2*real(i)/N - 1
          x(0,2) = 2*real(j)/N - 1
          do k=1, M+1  ! layer
            do q=1,2 ! row
              x(k,q) = tanh(( &
                x(k-1,1)*w(k,q,1)+x(k-1,2)*w(k,q,2)+b(k,q) &
                )/T)
            end do
          end do
          p(i,j) = x(M+1,1)
        end do
      end do
      write(str,*) "set title 't=",ii,"; M=",M,"'"
      call gnuplotExecute(str)
      write (str, *) "splot '-' "// &
      " binary array=",N+1,"x",N+1," dx=",2./(N), &
      " origin=(-1,-1,0) format='%float' ", &
      " with image , '-' w p pt 7 ps 5 lc palette", char(0)
      call gnuplotExecute(str)
      call gnuplotWrite(P, sizeof(p))
      do kk=1,NE
        write(str, *) xx(kk,1), xx(kk,2), yy(kk), char(0)
        call gnuplotExecute(str)
      end do
      call gnuplotExecute("end")
      call gnuplotFlush()

      !pause
    end if
  end do
  print *, ii, error
```

```
  end do
  pause
  call gnuplotClose()
end program
```

Unsupervised learning

- In unsupervised learning, examples are classified using the correlations among them.
- One popular implementation is the Kauhonen competitive network.
- still to be done...

Attractor neural networks

- Higher brain functions (the cortex) is better represented by a highly connected network.
- The Hopfield model is a symmetric, fully connected network.
- The Hebbian rule allows to store patterns ($\xi^{(k)}$):

$$w_{ij} = \sum_k \xi_i^{(k)} \xi_j^{(k)}$$

- still to be done...