Session #09&10

CMSC 409: Artificial Intelligence

http://www.people.vcu.edu/~mmanic/

Virginia Commonwealth University,
Fall 2023,
Dr. Milos Manic
(mmanic@vcu. edu)

1

CMSC 409: Artificial Intelligence

Session # 09&10

Topics for today

- Announcements
- Previous session review
- Project 1 review, lessons learned
- Perceptron learning agent
  - *Perceptron learning*
  - *Learning example*
  - *Graphical illustration*
    - *Learning constant & hard activation function*
  - *Hard vs. soft activation*

2

CMSC 409: Artificial Intelligence

Session # 09&10

Topics for today (cont.)

- Selecting rectangular area
  - *Architecture*
  - *Matlab code*
  - *The effect of gain on control surface (soft activation function)*
- Selecting any two-dimensional area
  - *Net/Out values by neurons & layers and gain effect*
  - *Partitioning in 3-dim space*
- Learning - the effect of transfer function

CMSC 409: Artificial Intelligence

Announcements Session # 09&10

- Canvas
  - *New slides posted*
- Office hours zoom
  - *Zoom disconnects me after 45 mins of inactivity. Feel free to chat me via zoom if that happens and I will reconnect (zoom chat welcome outside of office hours as well)!*
- Project #2
  - *Deadline Oct. 3 (noon)*
- Paper (optional)
  - *The 2nd draft due Oct. 10 (noon)*
  - *Literature review and updated problem description (check out the class paper instructions for the 2nd draft)*
- Subject line and signature
  - *Please use [CMSC 409] Last_Name Question*

Project 1 review

lessons learned

5

Current standing (Pr.1)

STATISTICS

Count	27
Minimum Value	0
Maximum Value	14
Range	14
Average	9.88
Median	11
Standard Deviation	3.71
Variance	13.79

Note: per student (multiple students in one group)

GRADE DISTRIBUTION

Greater than 100	0
90 - 100	1
80 - 89	12
70 - 79	3
60 - 69	5
50 - 59	2
40 - 49	1
30 - 39	0
20 - 29	0
10 - 19	1
0 - 9	2
Less than 0	0

6

Project 1 review

Part A

- Data should be normalized
- The goal was to estimate decision line manually (by hand)
- Plot should be included in the report
- Axis of the plot
  - should be named
  - units should be marked
  - For ex: weight [lb], cost [USD]
- Definition of the neuron should be included (net & out)
- Artificial counterpart of biological neuron decision (separation) line definition
  - Unit functionality/inequality, for ex. ax + by + c >= 0, should be written out; weights and bias should be stated
  - Threshold = -bias
  - (ax + by + c = 0 or ax + by + c < 0 not uniquely defining neuron)

Project 1 review (cont.)

- Fields of the confusion matrix should be stated (not only the definition)
- The rates were requested (TP, FP, TN, FN):
  - *ACC (accuracy), Recall or True Positive rate (TP), False Positive rate (FP), True Negative rate (TN), False Negative rate (FN), Precision (P):*

$$ACC(M) = \frac{a+d}{a+b+c+d} \quad TP(M) = \frac{a}{a+b} \quad FP(M) = \frac{c}{c+d}$$

$$TN(M) = \frac{d}{c+d} \quad FN(M) = \frac{b}{a+b} \quad P(M) = \frac{a}{a+c}$$

- State the equations, and steps during calculations

Submission

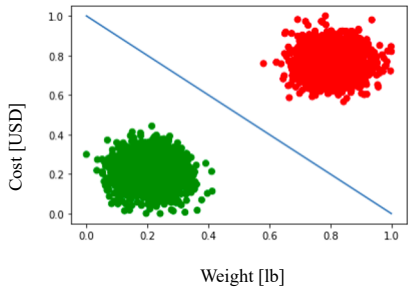
- Deliverable
  - Send the report in the pdf format
  - All deliverables should be included in the zip file (Do not attach report separately)
- Naming of zip file
  - Name the zip file as asked in the Project specification
- Team work:
  - Clearly state the workload distribution

Project 1, solution

Part A, dataset A

A) 1. 2.

1. Normalize the data, then plot the data (for two vehicle types)
2. Manually draw (by hand) a separation line. This will be a linear which separates big and small cars.



Linear Separator: y = -x + 1

Part A, dataset A

A) 1.3

3. Determine the equation of this linear separator
   a. Write the definition of a neuron – this will be an inequality (Note: think of the inequalities we covered in class).
   b. Determine the weights and threshold. Comment.

$$net = \sum_{i=1}^{n} w_i x_i + w_{n+1} \qquad out = \begin{cases} 1 & if\ net \geq 0 \\ 0 & if\ net < 0 \end{cases}$$

Inequality: $x + y - 1 \geq 0$

Weights: wx = 1, wy = 1, wb = - 1

Threshold: T = -wb = 1

Part A, dataset A

A) 1.4 & 1.5

a.) True Positive (Big classified as big) = 2000
b.) False Negative (Big classified as small) = 0
c.) False Positive (Small classified as big) = 0
d.) True Negative (Small classified as small) = 2000

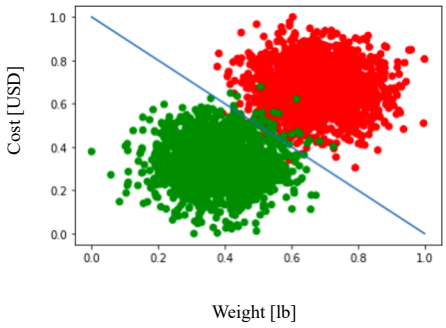
$$ACC(M) = \frac{a+d}{a+b+c+d} \quad TP(M) = \frac{a}{a+b} \quad FP(M) = \frac{c}{c+d}$$
$$TN(M) = \frac{d}{c+d} \quad FN(M) = \frac{b}{a+b} \quad P(M) = \frac{a}{a+c}$$

ACC = 1
Error = 0
TPR = 1
FPR = 0
TNR = 1
FNR = 0

13

Part A, dataset B

A) 1. 2.



Linear separator: y = -x + 1

14

Part A, dataset B

A) 1.3

$$net = \sum_{i=1}^{n} w_i x_i + w_{n+1} \qquad out = \begin{cases} 1 & if\ net \geq 0 \\ 0 & if\ net < 0 \end{cases}$$

Inequality: x + y - 1 ≥ 0

Weights: wx = 1, wy = 1, wb = - 1

Threshold: T = -wb = 1

15

Part A, dataset B

A) 1.4 & 1.5

True Positive (Big classified as big) = 1968
False Negative (Big classified as small) = 18
False Positive (Small classified as big) = 32
True Negative (Small classified as small) = 1982

$$ACC(M) = \frac{a+d}{a+b+c+d} \qquad TP(M) = \frac{a}{a+b} \qquad FP(M) = \frac{c}{c+d}$$

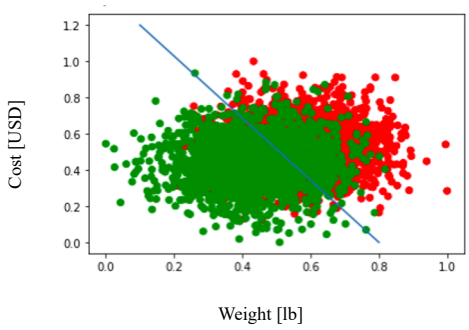
$$TN(M) = \frac{d}{c+d} \qquad FN(M) = \frac{b}{a+b} \qquad P(M) = \frac{a}{a+c}$$

ACC = 0.9875
Error = 0.0125
TPR = 0.991
FPR = 0.016
TNR = 0.984
FNR = 0.009

16

Part A, dataset C

A) 1. 2.



Linear Separator: y = -1.41x + 1.30

17

Part A, dataset C

A) 1.3

$$net = \sum_{i=1}^{n} w_i x_i + w_{n+1} \qquad out = \begin{cases} 1 & if\ net \geq 0 \\ 0 & if\ net < 0 \end{cases}$$

Inequality: $1.41x + y - 1.3 \geq 0$

Weights: wx = 1.41, wy = 1, wb = -1.3

Threshold: T = -wb = 1.3

18

Part A, dataset C

A) 1.4 & 1.5

True Positive (Big classified as big) = 1441
False Negative (Big classified as small) = 539
False Positive (Small classified as big) = 559
True Negative (Small classified as small) = 1461

$$ACC(M) = \frac{a+d}{a+b+c+d} \quad TP(M) = \frac{a}{a+b} \quad FP(M) = \frac{c}{c+d}$$
$$TN(M) = \frac{d}{c+d} \quad FN(M) = \frac{b}{a+b} \quad P(M) = \frac{a}{a+c}$$

ACC = 0.7255
Error = 0.2745
TPR = 0.7277
FPR = 0.2767
TNR = 0.7232
FNR = 0.2722

Part B

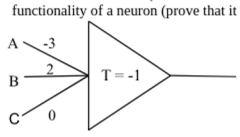
PROBLEM:

B) McCulloch-Pitts neurons (5 pts)

3. Create a truth table for the artificial neuron below. What is the **functionality of** this neuron? (3 points)
4. Given the same set of weights and the determined functionality of a neuron, what would be the **range of possible values for threshold**? (2 points)

Note: Consider unipolar hard threshold activation function (possible inputs/outputs are obviously 0 & 1). Always start with the unit definition (net, output). The functionality of the neuron is a Boolean function.

Hint: The truth table (similar to the one in class) should present inequalities that will evidence the functionality of a neuron (prove that it works as promised).



SOLUTION:

$$net = \sum_{i=1}^{n} w_i x_i \qquad out = \begin{cases} 1 & if\ net \geq 0 \\ 0 & if\ net < 0 \end{cases}$$

A	B	C	A'+B	Inequality	Sum	out
0	0	0	1	0 >= T ; T =-1	0	0 >=- 1
0	0	1	1	Wc >= T	0	0 >= -1
0	1	0	1	Wb >= T	2	2 >= -1
0	1	1	1	Wb + Wc >= T	2	2 >= -1
1	0	0	0	Wa < T	-3	-3 < -1
1	0	1	0	Wa + Wc < T	-3	-3 < -1
1	1	0	1	Wa + Wb >=T	-1	-1 >= -1
1	1	1	1	Wa + Wb + Wc >= T	-1	-1 >= -1

From here, the "decision line" or a neuron is defined as:
-3*A+2*B>=0

*3. Create a truth table for the artificial neuron below. What is the **functionality of** this neuron? (3 points)*
The functionality is A'+B
Note: 'A+B means: NOT(A) OR B

*4. Given the same set of weights and the determined functionality of a neuron, what would be the **range of possible values for threshold**? (2 points)*
The valid range of threshold values is: (-3, -1]

Part B

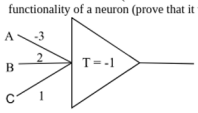
PROBLEM:

B) McCulloch-Pitts neurons (5 pts)

1. Create a truth table for the artificial neuron below. What is the **functionality of** this neuron? (3 points)
2. Given the same set of weights and the determined functionality of a neuron, what would be the **range of possible values for threshold**? (2 points)

Note: Consider unipolar hard threshold activation function (possible inputs/outputs are obviously 0 & 1). Always start with the unit definition (net, output). The functionality of the neuron is a Boolean function.

Hint: The truth table (similar to the one in class) should present inequalities that will evidence the functionality of a neuron (prove that it works as promised).



SOLUTION:

$$net = \sum_{i=1}^{n} w_i x_i \qquad out = \begin{cases} 1 & if\ net \geq 0 \\ 0 & if\ net < 0 \end{cases}$$

A	B	C	A'+B	Inequality	Sum	out
0	0	0	1	0 >= T ; T =-1	0	0 >=- 1
0	0	1	1	Wc >= T	1	1 >= -1
0	1	0	1	Wb >= T	2	2 >= -1
0	1	1	1	Wb + Wc >= T	3	3 >= -1
1	0	0	0	Wa < T	-3	-3 < -1
1	0	1	0	Wa + Wc < T	-2	-2 < -1
1	1	0	1	Wa + Wb >=T	-1	-1 >= -1
1	1	1	1	Wa + Wb + Wc >= T	0	0 >= -1

From here, the "decision line" or a neuron is defined as:
-3*A+2*B+1>=0

*1. Create a truth table for the artificial neuron below. What is the **functionality of** this neuron? (3 points)*

The functionality is A'+B
Note: 'A+B means: NOT(A) OR B

*2. Given the same set of weights and the determined functionality of a neuron, what would be the **range of possible values for threshold**? (2 points)*

The valid range of threshold values is: (-2 to -1]

21

Perceptron

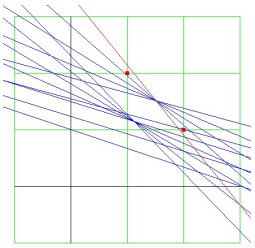
22

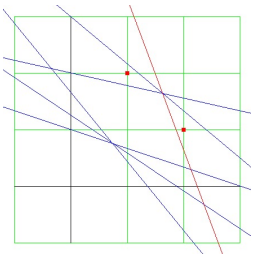
Perceptron learning

- *Perceptron learning*
- *Learning example*
- *Graphical illustration*
  - *Learning constant & hard activation function*
- *Learning example in Perl*
- *Hard vs. soft activation function*
  - *Soft activation function*

23

This was hard threshold function…

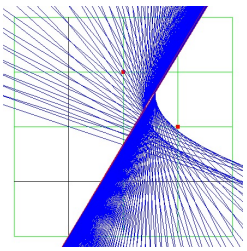


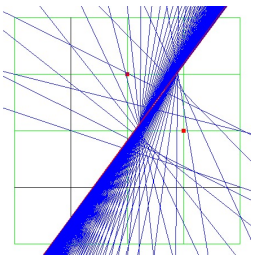


learning $\alpha = 0.1$

… how about some other?

learning $\alpha = 0.3$





24

Perceptron training in PERL (without graphics)

Perl

```
$ite=8;                 # number of training cycles
$np=2;                  # number of patterns
$ni=3;                  # number of augmented inputs
$alpha=0.1;             # learning constant
@ww= (1, 3, -3);  # array of weights
@pat=([1, 2, 1],[2, 1, 1]);    # patterns as 2-dim array
@dout=(-1, 1);      # desired output as 1-dim array
```

```
for $n (0..$ite-1) {            # number of training cycles
        for $p (0..$np-1) {# for all patterns
        $net=0;
        for $i (0..$ni-1) {  # for all inputs
        $net = $net +$ww[$i]*$pat[$p][$i];
                }
                $ou[$p] = sign($net);          # use activation function
                $err=$dout[$p]-$ou[$p];
                $learn=$alpha*$err;
                &printdata($n,$p,$net,$err,$learn,@ww);
                for $i (0..$ni-1) { # for all weights
        $ww[$i] = $ww[$i] + $learn*$pat[$p][$i];  # change weights
                }
        }
}
```


25

Perceptron training in Python (without graphics)

Python

```
ite=8;  # number of training cycles
np=2; # number of patterns
ni=3; # number of augmented inputs
alpha=0.1; # learning constant
ww= [1, 3, -3];  # array of weights
pat=([1, 2, 1],[2, 1, 1]); # patterns as 2-dim array
dout=(-1, 1);      # desired output as 1-dim array
```

```
#print(type(ww))
for iteration in range (0, ite): # number of training cycles
    ou = [0,0] # temporary array to store the output
    for pattern in range (0, np): # for all patterns
        net = 0
        for i in range(0, ni): # for all inputs
            net = net + ww[i]*pat[pattern][i]

        ou[pattern] = sign(net); # use activation function
        err = dout[pattern] - ou[pattern];
        learn = alpha*err;
        printdata(iteration, pattern, net, err, learn, ww);
        for i in range(0, ni): # for all inputs
            ww[i] = ww[i] + learn*pat[pattern][i]
```


26

Perceptron training in Python (Jupyter notebook)

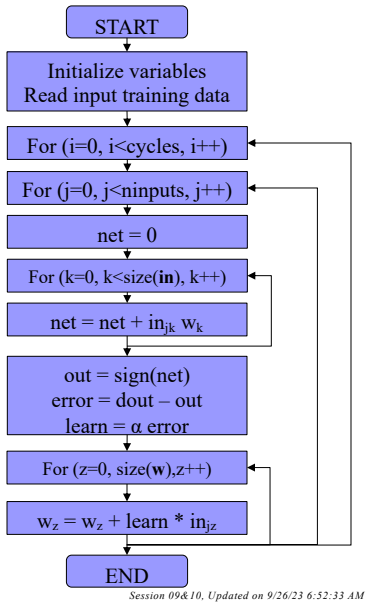
Python

```
def sign(net):
    if net >= 0:
        return 1
    else:
        return -1

def printdata(iteration, pattern, net, err, learn, ww):
    #ite=  0 p=0 net= 4.00 err =-2.000 lrn =-0.200 weights:  1.00   3.00  -3.00
    ww_formated = [ '%.2f' % elem for elem in ww ]
    print("ite= ", iteration, ' pat=', pattern, ' net=', round(net,5) , ' err=', err, ' lrn=', learn,
          ' weights=', ww_formated )

ite=8;  #number of training cycles
np=2; #number of patterns
ni=3; #number of augmented inputs
alpha=0.1; #learning constant
ww= [1, 3, -3];  #array of weights
pat=([1, 2, 1],[2, 1, 1]); #patterns as 2-dim array
dout=(-1, 1);      #desired output as 1-dim array
#print(type(ww))
for iteration in range (0, ite): # number of training cycles

    ou = [0,0] # temporary array to store the output
    for pattern in range (0, np): # for all patterns
        net = 0
        for i in range(0, ni): # for all inputs
            net = net + ww[i]*pat[pattern][i]

        ou[pattern] = sign(net); #use activation function
        err = dout[pattern] - ou[pattern];
        learn = alpha*err;
        printdata(iteration, pattern, net, err, learn, ww);
        for i in range(0, ni): # for all inputs
            #print(ww[i]+1)
            ww[i] = ww[i] + learn*pat[pattern][i]
```

27

Perceptron training, steps

- cycles → Number of training cycles
- ninputs → Number of (augmented) inputs
- α → Learning constant
- **w** → Array of weights
- **in** → Array of input
- dout → Desired output



28

Training one neuron with soft activation function

$out = \tanh(k*net)$ *learning constant:* $\alpha = 0.1$

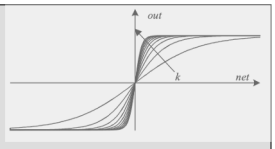
$ou[$p] = sign($net);

replaced by: $ou[$p] = fbip($net);

Perl

where fbip is:

```
sub fbip
{
        my $k=0.2;
        my $x = $_[0];    #  or $x = shift(@_);
        my $r= 2/(1+exp(-2*$k*$x))-1;
return $r;
}
```

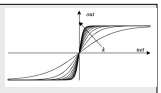


29

Training one neuron with soft activation function

Perl

```
sub fbip
{
        my $k=0.2;          # gain in neuron
        my $x = $_[0];    #  or $x = shift(@_);
        my $r= 2/(1+exp(-2*$k*$x))-1;
return $r;
}
```



or shorter:

```
sub fbip
{
        my $k=0.2;          # gain in neuron
        2/(1+exp(-2*$k*$_[0]))-1;
}
```


30

Program in Python (fbip)

Python

```
import math

def sign(net):
    if net >= 0:
        return 1
    else:
        return -1

def fbip(net):
    k = 0.2
    return 2/( 1+math.exp(-2*k*net) ) - 1

def printdata(iteration, pattern, net, err, learn, ww):
    #ite=  0 p=0 net= 4.00 err =-2.000 lrn =-0.200 weights:  1.00   3.00  -3.00
    ww_formated = [ '%.5f' % elem for elem in ww ]
    print("ite= ", iteration, ' pat=', pattern, ' net=', round(net,5) , ' err=', err, ' lrn=', learn,
          ' weights=', ww_formated )

ite=8;  ----*----*----# number of training cycles
np=2;  ---*----*----# number of patterns
ni=3;  ---*----*----# number of augmented inputs
alpha=0.1; *----*----# learning constant
ww= [1, 3, -3];  ----*----*----# array of weights
pat=([1, 2, 1],[2, 1, 1]); -# patterns as 2-dim array
dout=(-1, 1);       *----*----# desired output as 1-dim array
#print(type(ww))
for iteration in range (0, ite): # number of training cycles

    ou = [0,0] # temporary array to store the output
    for pattern in range (0, np): # for all patterns
        net = 0
        for i in range(0, ni): # for all inputs
            net = net + ww[i]*pat[pattern][i]

        ou[pattern] = fbip(net); ----# use activation function
        err = dout[pattern] - ou[pattern];
        learn = alpha*err;
        printdata(iteration, pattern, net, err, learn, ww);
        for i in range(0, ni): # for all inputs
            #print(ww[i]+1)
            ww[i] = ww[i] + learn*pat[pattern][i]
```

© M

31

Training one neuron with soft activation function

Python

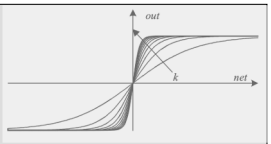
$out = \tanh(k*net)$ *learning constant:* $\alpha = 0.1$

ou[pattern] = sign(net);

replaced by: ou[pattern] = fbip(net);

where fbip is:

```
def fbip(net):

   k = 0.2

   return 2/( 1+math.exp(-2*k*net) ) - 1
```



32

Perceptron learning

- *Perceptron learning*
- *Learning example*
- *Graphical illustration*
  - *Learning constant & hard activation function*
- *Learning example in Perl*
- *Hard vs. soft activation function*
  - *Soft activation function*

33

Training one neuron with soft activation function

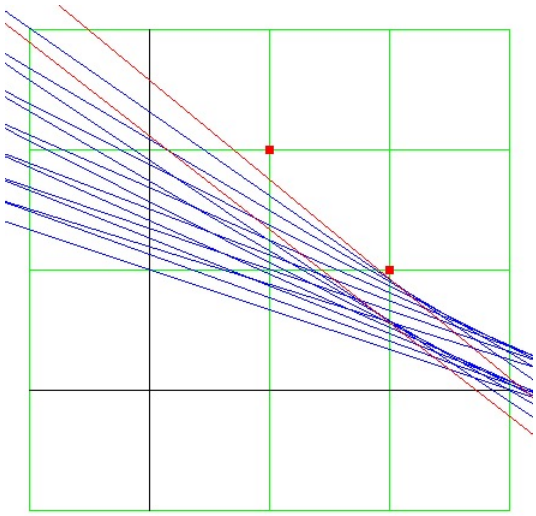
alpha=0.1

ite= 0 p=0 net= 4.00	err =-1.664	lrn =-0.166 weights: 1.00 3.00 -3.00	
ite= 0 p=1 net= 1.17	err = 0.771	lrn = 0.077 weights: 0.83 2.67 -3.17	TE= 3.362784
ite= 1 p=0 net= 3.39	err =-1.590	lrn =-0.159 weights: 0.99 2.74 -3.09	
ite= 1 p=1 net= 0.84	err = 0.834	lrn = 0.083 weights: 0.83 2.43 -3.25	TE= 3.223804
ite= 2 p=0 net= 2.85	err =-1.515	lrn =-0.152 weights: 1.00 2.51 -3.16	
ite= 2 p=1 net= 0.58	err = 0.885	lrn = 0.088 weights: 0.84 2.21 -3.32	TE= 3.079336
ite= 3 p=0 net= 2.38	err =-1.444	lrn =-0.144 weights: 1.02 2.30 -3.23	
ite= 3 p=1 net= 0.39	err = 0.923	lrn = 0.092 weights: 0.88 2.01 -3.37	TE= 2.935209
ite= 4 p=0 net= 1.98	err =-1.376	lrn =-0.138 weights: 1.06 2.10 -3.28	
ite= 4 p=1 net= 0.25	err = 0.949	lrn = 0.095 weights: 0.92 1.82 -3.42	TE= 2.795590
ite= 5 p=0 net= 1.63	err =-1.314	lrn =-0.131 weights: 1.11 1.92 -3.32	
ite= 5 p=1 net= 0.17	err = 0.967	lrn = 0.097 weights: 0.98 1.66 -3.45	TE= 2.662808
ite= 6 p=0 net= 1.32	err =-1.258	lrn =-0.126 weights: 1.18 1.75 -3.36	
ite= 6 p=1 net= 0.12	err = 0.977	lrn = 0.098 weights: 1.05 1.50 -3.48	TE= 2.537752
ite= 7 p=0 net= 1.06	err =-1.208	lrn =-0.121 weights: 1.24 1.60 -3.39	
ite= 7 p=1 net= 0.10	err = 0.980	lrn = 0.098 weights: 1.12 1.36 -3.51	TE= 2.420410

$$TE = \sum_{p=1}^{P} (d_p - o_p)^2$$

34

Training one neuron with soft activation function



learning $\alpha = 0.1$

35

Training one neuron with soft activation function

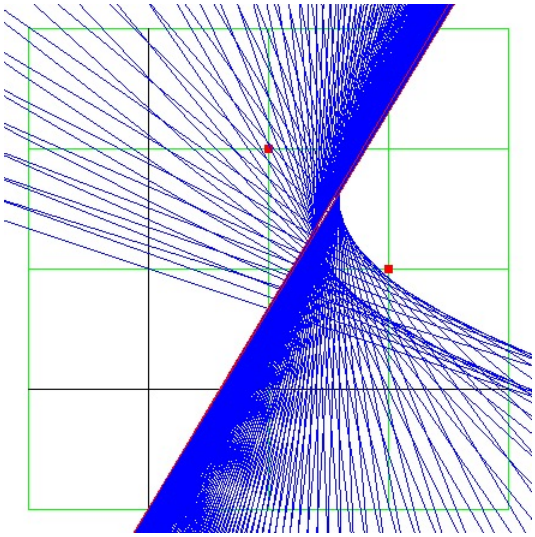
alpha=0.1

$$TE = \sum_{p=1}^{P} (d_p - o_p)^2$$

……..	…….	……..	
ite= 91 p=0 net=-4.44	err =-0.290	lrn =-0.029 weights: 5.72 -3.31 -3.53	……..
ite= 91 p=1 net= 4.45	err = 0.288	lrn = 0.029 weights: 5.69 -3.37 -3.56	TE= 0.167104
ite= 92 p=0 net=-4.47	err =-0.287	lrn =-0.029 weights: 5.75 -3.34 -3.53	
ite= 92 p=1 net= 4.48	err = 0.285	lrn = 0.029 weights: 5.72 -3.40 -3.56	TE= 0.163746
ite= 93 p=0 net=-4.50	err =-0.284	lrn =-0.028 weights: 5.78 -3.37 -3.53	
ite= 93 p=1 net= 4.51	err = 0.283	lrn = 0.028 weights: 5.75 -3.43 -3.56	TE= 0.160483
ite= 94 p=0 net=-4.53	err =-0.281	lrn =-0.028 weights: 5.81 -3.40 -3.53	
ite= 94 p=1 net= 4.54	err = 0.280	lrn = 0.028 weights: 5.78 -3.46 -3.56	TE= 0.157312
ite= 95 p=0 net=-4.56	err =-0.278	lrn =-0.028 weights: 5.83 -3.43 -3.53	
ite= 95 p=1 net= 4.57	err = 0.277	lrn = 0.028 weights: 5.81 -3.48 -3.56	TE= 0.154231
ite= 96 p=0 net=-4.58	err =-0.276	lrn =-0.028 weights: 5.86 -3.46 -3.53	
ite= 96 p=1 net= 4.60	err = 0.274	lrn = 0.027 weights: 5.83 -3.51 -3.56	TE= 0.151235
ite= 97 p=0 net=-4.61	err =-0.273	lrn =-0.027 weights: 5.89 -3.48 -3.53	
ite= 97 p=1 net= 4.63	err = 0.272	lrn = 0.027 weights: 5.86 -3.54 -3.56	TE= 0.148322
ite= 98 p=0 net=-4.64	err =-0.270	lrn =-0.027 weights: 5.92 -3.51 -3.53	
ite= 98 p=1 net= 4.65	err = 0.269	lrn = 0.027 weights: 5.89 -3.57 -3.56	TE= 0.145488
ite= 99 p=0 net=-4.67	err =-0.268	lrn =-0.027 weights: 5.94 -3.54 -3.53	
ite= 99 p=1 net= 4.68	err = 0.267	lrn = 0.027 weights: 5.92 -3.59 -3.56	TE= 0.142732

36

Training one neuron with soft activation function

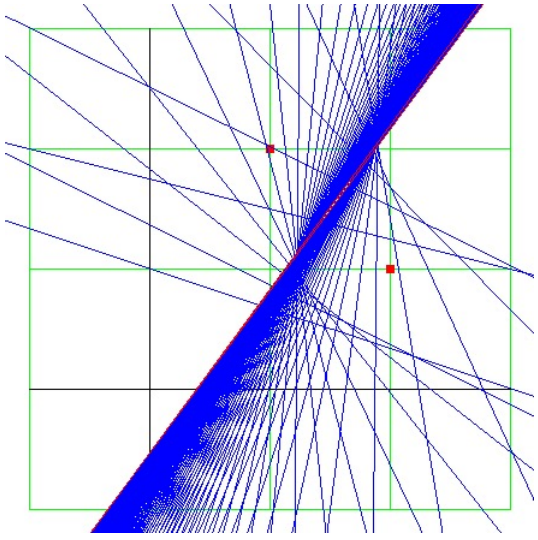


learning constant:

$$\alpha = 0.1$$

37

Training one neuron with soft activation function



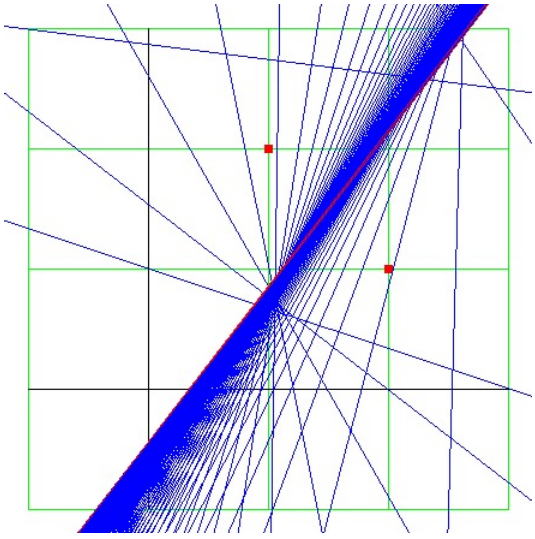
learning constant:

$$\alpha = 0.3$$

Nice solution!

38

Training one neuron with soft activation function



learning constant:

$$\alpha = 0.5$$

Almost optimal solution!

However, for larger alpha, jumps too big, process oscillates and never converges!!!

39

Training one neuron with soft activation function

alpha=0.3

$$TE = \sum_{p=1}^{P} (d_p - o_p)^2$$

……..	…….	……..	
ite= 91 p=0 net=-7.53	err =-0.094	lrn =-0.028 weights: 8.81 -6.41 -3.53	……..
ite= 91 p=1 net= 7.55	err = 0.093	lrn = 0.028 weights: 8.79 -6.46 -3.56	TE= 0.017442
ite= 92 p=0 net=-7.56	err =-0.093	lrn =-0.028 weights: 8.84 -6.44 -3.53	
ite= 92 p=1 net= 7.58	err = 0.092	lrn = 0.028 weights: 8.81 -6.49 -3.56	TE= 0.017063
ite= 93 p=0 net=-7.59	err =-0.092	lrn =-0.027 weights: 8.87 -6.46 -3.53	
ite= 93 p=1 net= 7.61	err = 0.091	lrn = 0.027 weights: 8.84 -6.52 -3.56	TE= 0.016696
ite= 94 p=0 net=-7.62	err =-0.091	lrn =-0.027 weights: 8.90 -6.49 -3.53	
ite= 94 p=1 net= 7.63	err = 0.090	lrn = 0.027 weights: 8.87 -6.55 -3.56	TE= 0.016341
ite= 95 p=0 net=-7.65	err =-0.090	lrn =-0.027 weights: 8.92 -6.52 -3.53	
ite= 95 p=1 net= 7.66	err = 0.089	lrn = 0.027 weights: 8.90 -6.57 -3.56	TE= 0.015997
ite= 96 p=0 net=-7.67	err =-0.089	lrn =-0.027 weights: 8.95 -6.55 -3.53	
ite= 96 p=1 net= 7.69	err = 0.088	lrn = 0.026 weights: 8.92 -6.60 -3.56	TE= 0.015664
ite= 97 p=0 net=-7.70	err =-0.088	lrn =-0.026 weights: 8.98 -6.57 -3.53	
ite= 97 p=1 net= 7.72	err = 0.087	lrn = 0.026 weights: 8.95 -6.63 -3.56	TE= 0.015341
ite= 98 p=0 net=-7.73	err =-0.087	lrn =-0.026 weights: 9.00 -6.60 -3.53	
ite= 98 p=1 net= 7.74	err = 0.086	lrn = 0.026 weights: 8.98 -6.65 -3.56	TE= 0.015028
ite= 99 p=0 net=-7.76	err =-0.086	lrn =-0.026 weights: 9.03 -6.63 -3.53	
ite= 99 p=1 net= 7.77	err = 0.086	lrn = 0.026 weights: 9.00 -6.68 -3.56	TE= 0.014724

Note change in TE with change of alpha…

40

Perceptron learning

- *Perceptron learning*
- *Learning example*
- *Graphical illustration*
  - *Learning constant & hard activation function*
- *Learning example in Perl*
- *Hard vs. soft activation function*
  - *Soft activation function*

41

Training one neuron using the perceptron rule

(results obtained using perl program)

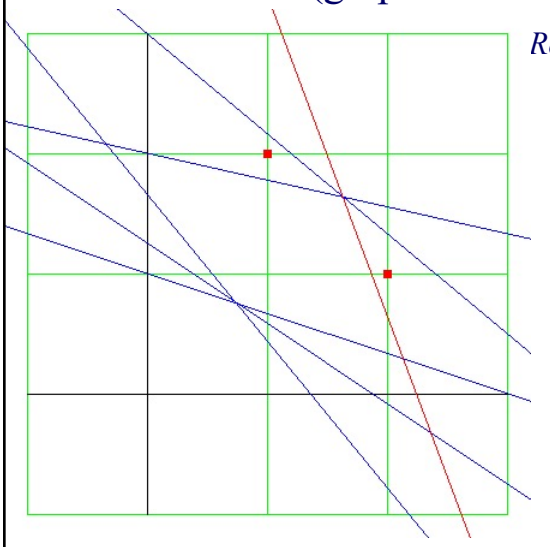
learning constant: $\alpha = 0.3$

```
ite=  0 p=0 net= 4.00 err =-2.000 lrn =-0.600 weights:  1.00   3.00  -3.00
ite=  0 p=1 net=-1.00 err = 2.000 lrn = 0.600 weights:  0.40   1.80  -3.60
ite=  1 p=0 net= 3.40 err =-2.000 lrn =-0.600 weights:  1.60   2.40  -3.00
ite=  1 p=1 net=-0.40 err = 2.000 lrn = 0.600 weights:  1.00   1.20  -3.60
ite=  2 p=0 net= 2.80 err =-2.000 lrn =-0.600 weights:  2.20   1.80  -3.00
ite=  2 p=1 net= 0.20 err = 0.000 lrn = 0.000 weights:  1.60   0.60  -3.60
ite=  3 p=0 net=-0.80 err = 0.000 lrn = 0.000 weights:  1.60   0.60  -3.60
ite=  3 p=1 net= 0.20 err = 0.000 lrn = 0.000 weights:  1.60   0.60  -3.60
```

Same example, same alpha, same results…

42

Training one neuron using the perceptron rule
(graphical illustration)



Results obtained using perl pgm.

Initial weight set & learning constant:

$$\mathbf{w} = [1 \quad 3 \quad -3]$$
$$\alpha = 0.3$$

Final weight set:

$$\mathbf{w} = [1.6 \quad 0.6 \quad -3.6]$$

...different alpha?

43

Training one neuron using the perceptron rule
(results obtained using perl program)

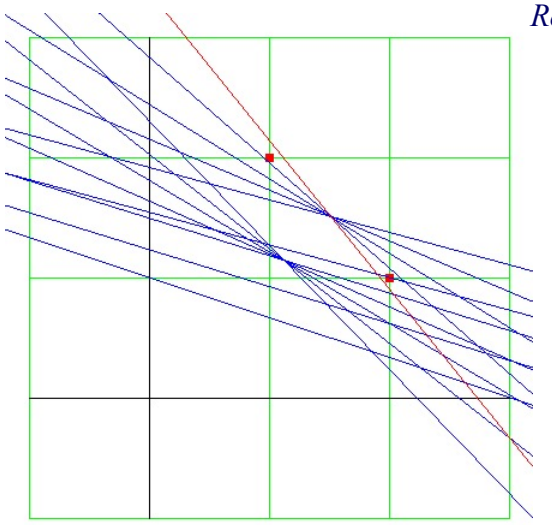
learning constant: $\alpha = 0.1$

```
ite=  0 p=0 net= 4.00 err =-2.000 lrn =-0.200 weights:  1.00   3.00  -3.00
ite=  0 p=1 net= 1.00 err = 0.000 lrn = 0.000 weights:  0.80   2.60  -3.20
ite=  1 p=0 net= 2.80 err =-2.000 lrn =-0.200 weights:  0.80   2.60  -3.20
ite=  1 p=1 net= 0.00 err = 0.500 lrn = 0.050 weights:  0.60   2.20  -3.40
ite=  2 p=0 net= 1.85 err =-2.000 lrn =-0.200 weights:  0.70   2.25  -3.35
ite=  2 p=1 net=-0.70 err = 2.000 lrn = 0.200 weights:  0.50   1.85  -3.55
ite=  3 p=0 net= 1.65 err =-2.000 lrn =-0.200 weights:  0.90   2.05  -3.35
ite=  3 p=1 net=-0.50 err = 2.000 lrn = 0.200 weights:  0.70   1.65  -3.55
ite=  4 p=0 net= 1.45 err =-2.000 lrn =-0.200 weights:  1.10   1.85  -3.35
ite=  4 p=1 net=-0.30 err = 2.000 lrn = 0.200 weights:  0.90   1.45  -3.55
ite=  5 p=0 net= 1.25 err =-2.000 lrn =-0.200 weights:  1.30   1.65  -3.35
ite=  5 p=1 net=-0.10 err = 2.000 lrn = 0.200 weights:  1.10   1.25  -3.55
ite=  6 p=0 net= 1.05 err =-2.000 lrn =-0.200 weights:  1.50   1.45  -3.35
ite=  6 p=1 net= 0.10 err = 0.000 lrn = 0.000 weights:  1.30   1.05  -3.55
ite=  7 p=0 net=-0.15 err = 0.000 lrn = 0.000 weights:  1.30   1.05  -3.55
```


44

Training one neuron using the perceptron rule (graphical illustration)

Results obtained using perl pgm.



Initial weight set & learning constant:

$$\mathbf{w} = [1 \quad 3 \quad -3]$$
$$\alpha = 0.1$$

Final weight set:

$$\mathbf{w} = [1.3 \quad 1.05 \quad -3.55]$$

45

Perceptron learning

- *Perceptron learning*
- *Learning example*
- *Graphical illustration*
  - *Learning constant & hard activation function*
- *Learning example in Perl*
- *Hard vs. soft activation function*
  - *Soft activation function*

46

Selecting an area

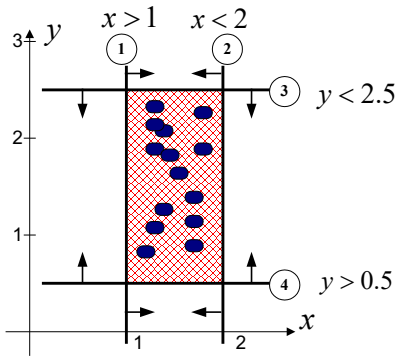
- *rectangular area*
- *arbitrary area*

47

- Selecting rectangular area
  - ***Architecture***
  - *Matlab code*
  - *The effect of gain on control surface (soft activation function)*
- Selecting any two-dimensional area
  - *Architecture*
  - *Net/Out values by neurons & layers and gain effect*

48

Selecting rectangular area with 4 partitions



neuron inequalities:

① $x-1>0$

② $-x+2>0$

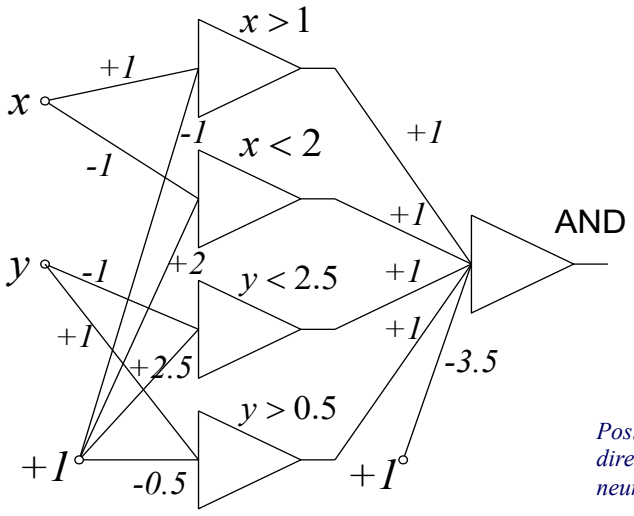
③ $-y+2.5>0$

④ $y-0.5>0$

Positive responses from adequate direction combined by AND neuron.

49

Selecting rectangular area with 4 partitions using neural network



neuron equations:

① $x-1>0$

② $-x+2>0$

③ $-y+2.5>0$

④ $y-0.5>0$

Positive responses from adequate direction combined by AND neuron.

50

- Selecting rectangular area
  - *Architecture*
  - ***Matlab code***
  - *The effect of gain on control surface (soft activation function)*
- Selecting any two-dimensional area
  - *Network architecture*
  - *Net/Out values by neurons & layers and gain effect*

MATLAB code for processing

```
n=51; k=10;
w1=[1 0 -1;-1 0  2; 0 -1 2.5; 0  1  -0.5];
for i= 1:n
     x(i)=3*(i-1)/(n-1);
     for j=1:n
          y(j)=3*(j-1)/(n-1);
          sum=0;
          for nn= 1:4
               net(nn)= w1(nn,1)*y(j)+w1(nn,2)*x(i)+w1(nn,3);
               o(nn)= 1/(1+exp(-k*net(nn)));
               sum=sum+o(nn);
          end;
     out(i,j)= 1/(1+exp(-k*(sum-3.5)));
     end;
end;
figure(1); clf; mesh(x,y,out); view(-20,50);
```

neuron equations:

① $x-1>0$

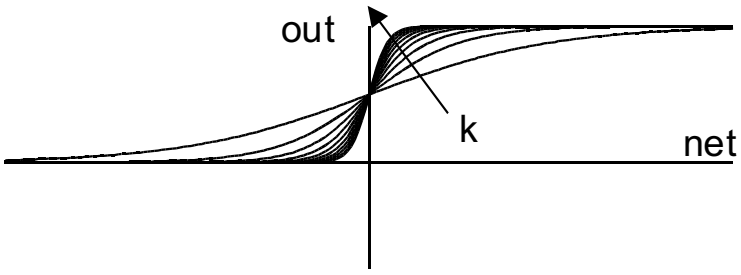
② $-x+2>0$

③ $-y+2.5>0$

④ $y-0.5>0$

- Selecting rectangular area
  - *Architecture*
  - *Matlab code*
  - ***The effect of gain on control surface (soft activation function)***
- Selecting any two-dimensional area
  - *Architecture*
  - *Net/Out values by neurons & layers*
  - *The effect of gain and activation function*

Activation functions

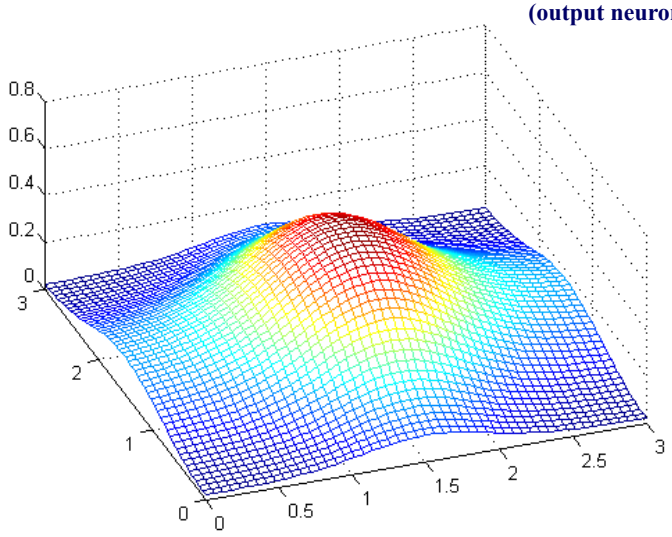


$$o = f(knet) = \frac{1}{1 + \exp(-knet)}$$

Soft activation function (larger k -> harder threshold)

Result for k=3 (neuron gain)

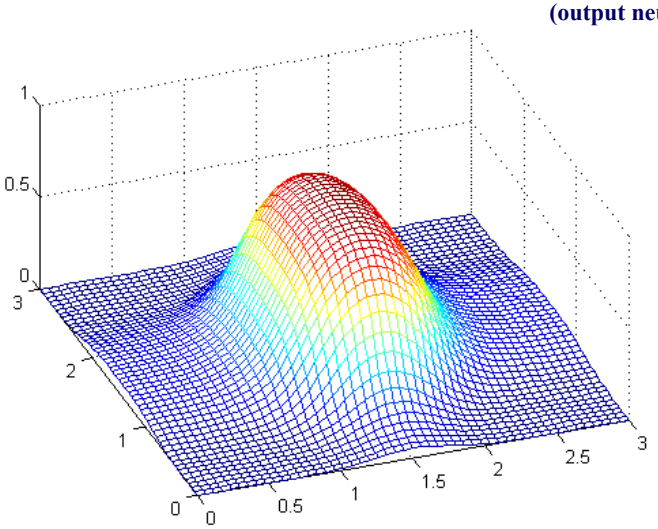
(output neuron only)



55

Result for k=5 (neuron gain)

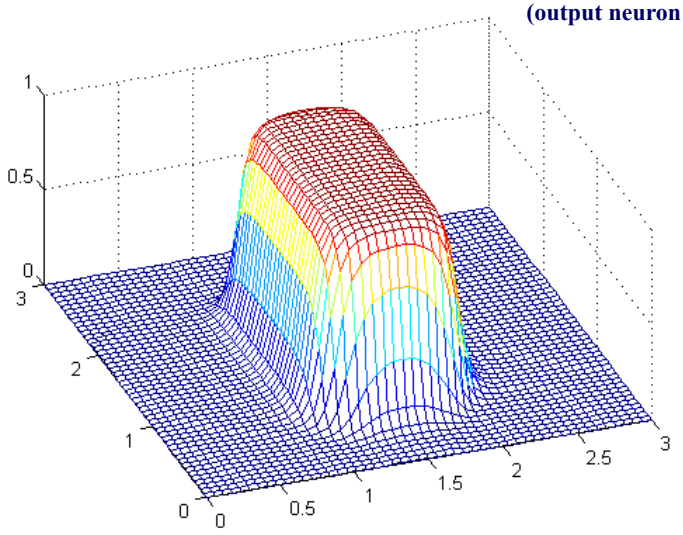
(output neuron only)



56

Result for k=10 (neuron gain)

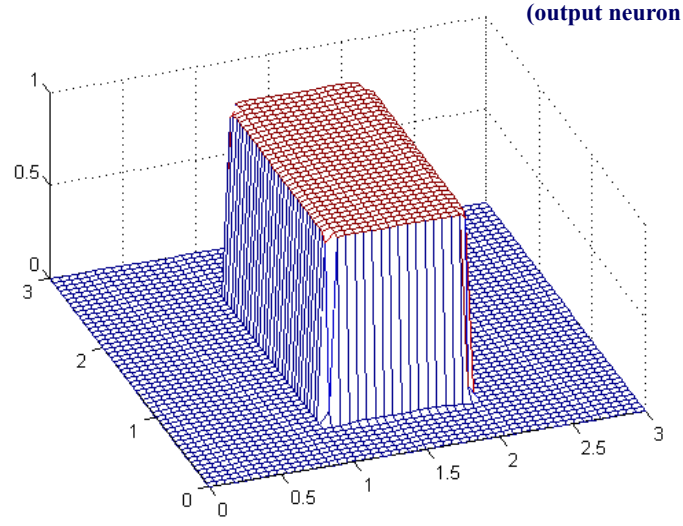
(output neuron only)



57

Result for k=30 (neuron gain)

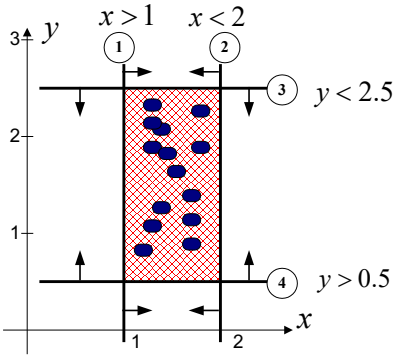
(output neuron only)



58

- Selecting rectangular area
  - *Architecture*
  - *Matlab code*
  - *The effect of gain on control surface (soft activation function)*
- Selecting any two-dimensional area
  - ***Architecture***
  - *Net/Out values by neurons & layers and gain effect*

Selecting rectangular area with 4 partitions



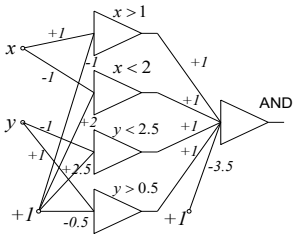
neuron equations:

① $x - 1 > 0$

② $-x + 2 > 0$

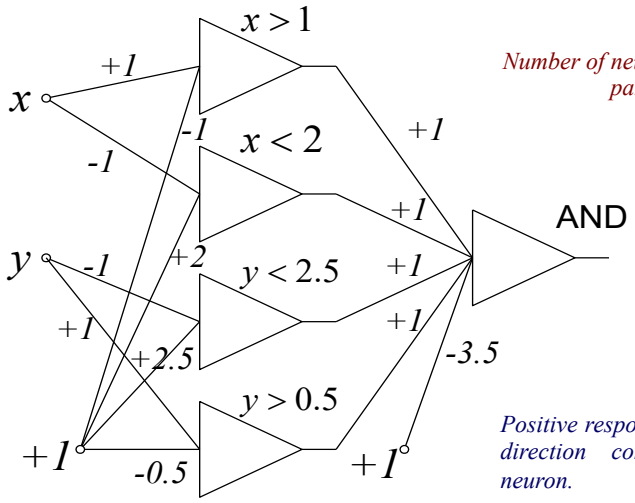
③ $-y + 2.5 > 0$

④ $y - 0.5 > 0$



Positive responses from adequate direction combined by AND neuron.

Selecting rectangular area with 4 partitions using neural network

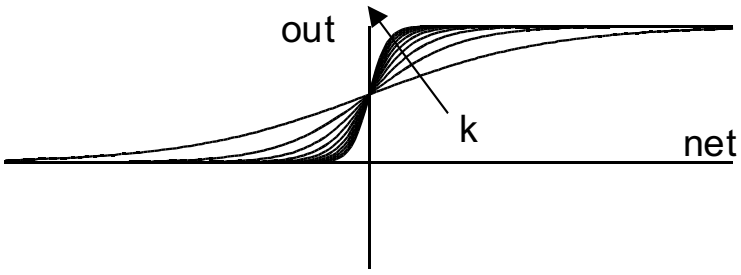


Number of neurons vs. number of partitions?

Positive responses from adequate direction combined by AND neuron.

61

Activation function

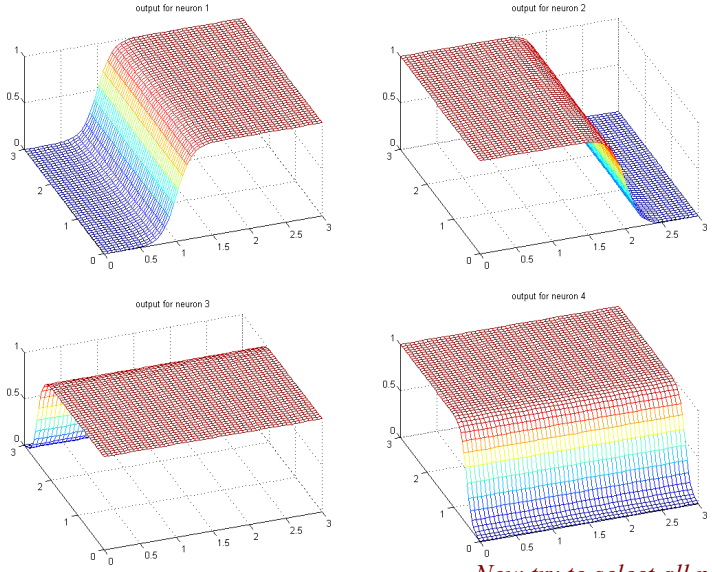


$$o = f(knet) = \frac{1}{1 + \exp(-knet)}$$

Soft activation function (larger k -> harder threshold)

62

Results for each neuron of 1st layer (gain=10)

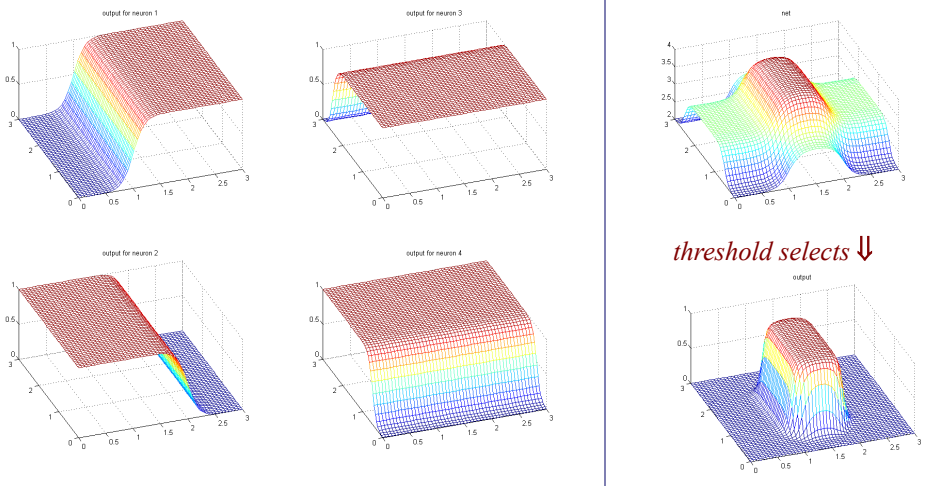


1. *Smooth transitions*
2. *Gain (code)*
3. *Larger k -> sharper surface*

Now try to select all positive outputs =>

63

Values for first and second layers (k=10)



threshold selects ⇓

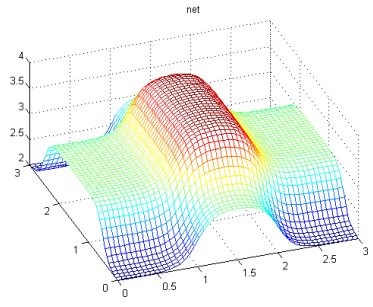
first layer

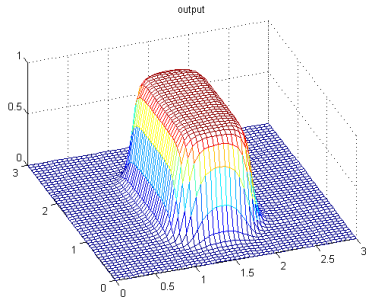
second layer

64

Net and output values for the output neuron (gain=10)

!





Net value

Output value

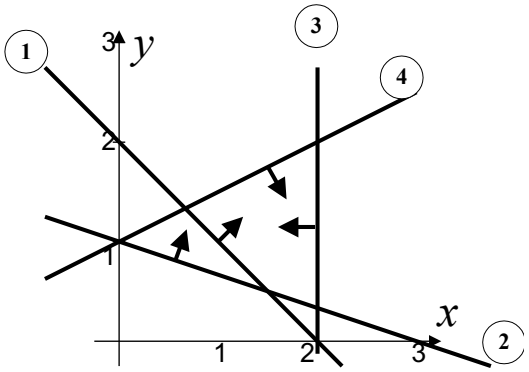
Only when all outputs positive!!

65

- Selecting rectangular area
  - *Architecture*
  - *Matlab code*
  - *The effect of gain on control surface (soft activation function)*
- Selecting any two-dimensional area
  - ***Architecture***
  - *Net/Out values by neurons & layers and gain effect*

66

Another area with 4 partitions



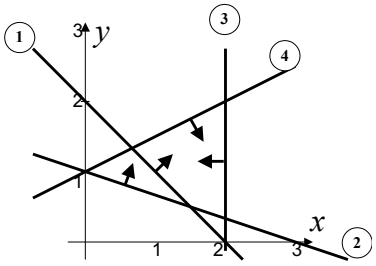
neuron equations

1. $\frac{x}{2}+\frac{y}{2}>1$
2. $\frac{x}{3}+\frac{y}{1}>1$
3. $\frac{x}{2}+\frac{y}{\infty}>1$
4. $\frac{x}{-2}+\frac{y}{1}<1$

Correct directions of each separation line (neuron)!

67

Another area with 4 partitions



neuron equations

1. $\frac{x}{2}+\frac{y}{2}>1$ $\quad x+y-2>0$
2. $\frac{x}{3}+\frac{y}{1}>1$ $\quad x+3y-3>0$
3. $\frac{x}{2}+\frac{y}{\infty}>1$ $\quad -x+0y+2>0$
4. $\frac{x}{-2}+\frac{y}{1}<1$ $\quad x-2y+2>0$

Weights in the first layer:

1 1 –2; 1 3 –3; – 1 0 +2; 1 –2 2;

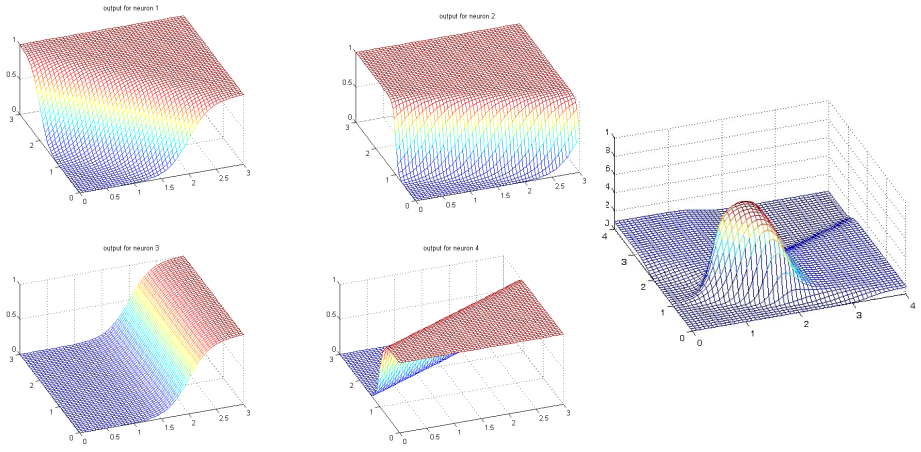
Weights in the second layer: 1 1 1 1 –3.5;

68

- Selecting rectangular area
  - *Architecture*
  - *Matlab code*
  - *The effect of gain on control surface (soft activation function)*
- Selecting any two-dimensional area
  - *Architecture*
  - ***Net/Out values by neurons & layers and gain effect***

Another area with 4 partitions

first layer **second layer**

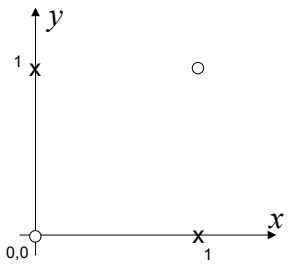


Non-linear mapping!!!

Partitioning in 3-dim space

XOR?

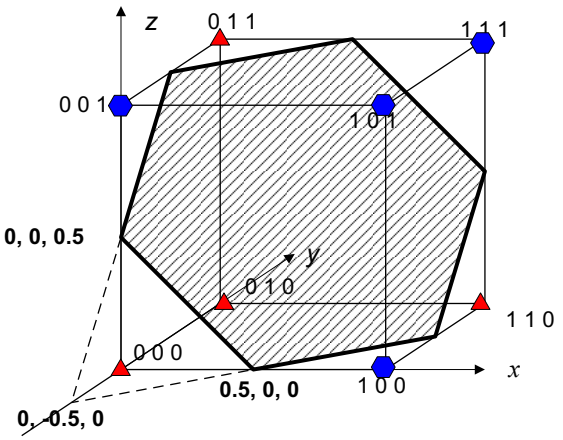
Linearly inseparable (one neuron)!



2 neurons (2 layers) YES!

Partitioning in 3-dim space

Partitioning set of cube vertices



Plane equation:

$$\frac{x}{0.5}+\frac{y}{-0.5}+\frac{z}{0.5}=1$$

or

$$x-y+z-0.5=0$$

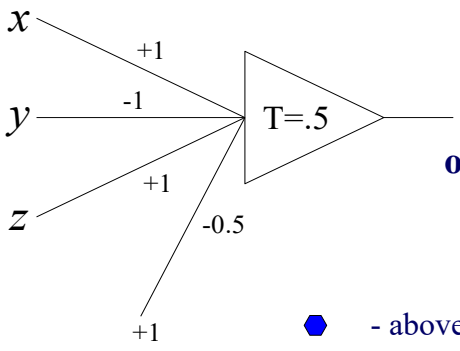
- above the plane
- below the plane

73

Partitioning in 3-dim space

A neuron partitioning a set of cube vertices

$$x-y+z-0.5\geq 0$$



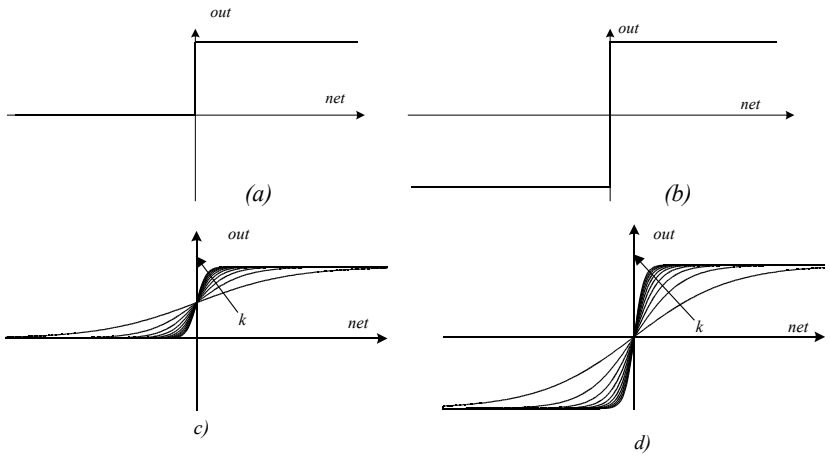
input	0 0 0 0 1 0 0 1 1 1 1 0	0 0 1 1 0 0 1 0 1 1 1 1
output	**0**	**1**

- above the plane
- below the plane

74

Learning Agents – transfer function

(for now, note the shape of the function only)



$$o = f(knet) = \frac{1}{1 + \exp(-knet)}$$

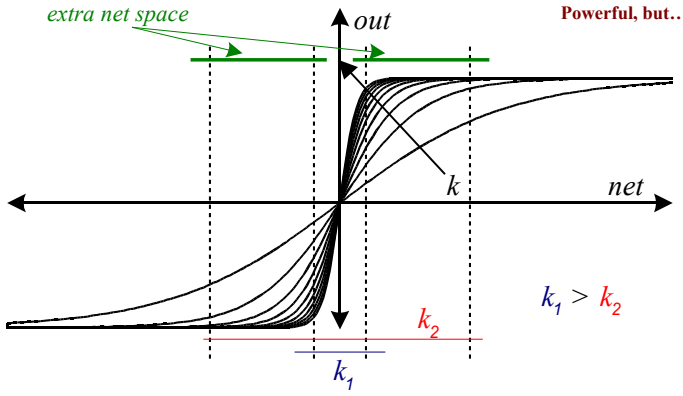
$$o = f(knet) = \tanh\left(\frac{knet}{2}\right) = \frac{1}{1 + \exp(-knet)} - 1$$

75

Learning - the effect of transfer function

76

Bipolar soft activation function, net vs. out



Powerful, but…vanishing gradient!

$$f_{bip}(k \cdot net) = \tanh(k \cdot net) = \frac{2}{1 + exp(-k \cdot net)} - 1$$

k grows larger => soft becomes hard activation function

k gets smaller => soft becomes linear activation function, y axis

77

Unipolar soft activation function, derivation

Unipolar soft activation function:

$$o_{uni} = f_{uni}(k \cdot net) = (\tanh(k \cdot net) + 1)/2 = \ldots$$

…which can be in couple of steps reduced to…

$$\ldots = \frac{exp(k \cdot net)}{exp(k \cdot net) + exp(-k \cdot net)} = \frac{1}{1 + exp(-2 \cdot k \cdot net)} =>$$

Finally: $$o_{uni} = f_{uni}(k \cdot net) = \frac{1}{1 + exp(-k \cdot net)}$$

First derivative:

$$f'_{uni}(k \cdot net) = \left((1 + exp(-k \cdot net))^{-1}\right)' = \ldots$$

…that after couple of steps…

$$\ldots = k \cdot \frac{exp(-k \cdot net) + 1 - 1}{(1 + exp(-k \cdot net))^2} = k \cdot \frac{1}{1 + exp(-k \cdot net)} \cdot \frac{exp(-k \cdot net) + 1 - 1}{1 + exp(-k \cdot net)} =>$$

Finally: $$f'_{uni}(k \cdot net) = k \cdot o \cdot (1 - o)$$

78

Bipolar soft activation function, derivation

Bipolar soft activation function:

$$f_{bip}(k \cdot net) = tanh(k \cdot net) = \frac{exp(k \cdot net) - exp(-k \cdot net)}{exp(k \cdot net) + exp(-k \cdot net)} =$$

$$= \frac{1 - exp(-2 \cdot k \cdot net)}{1 + exp(-2 \cdot k \cdot net)} = \ldots$$

which can be reduced through couple of manipulations to

$$\left(\frac{1 - exp(-2 \cdot k \cdot net)}{1 + exp(-2 \cdot k \cdot net)} = {}^{1-Y}/_{exp(-2 \cdot k \cdot net)+Y} = \frac{1 - Y + (1 + Y - 1 - Y)}{1 + Y} = \right.$$
$$\left. = \frac{2 + (-1 - Y)}{1 + Y} = \frac{2}{1 + Y} - 1 = \frac{2}{1 + exp(-2 \cdot k \cdot net)} - 1 \right)$$

$$\ldots = \frac{2}{1 + exp(-2 \cdot k \cdot net)} - 1$$

First derivative:

$$f'_{bip}(k \cdot net) = \left[tanh(k \cdot net) \right]' = \frac{4 \cdot k \cdot exp(-2 \cdot k \cdot net)}{(1 + exp(-2 \cdot k \cdot net))^2} = \ldots$$

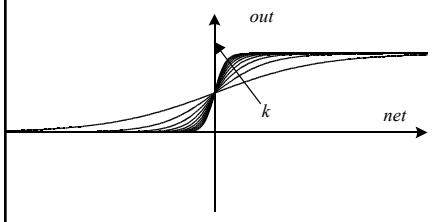
k in denominator grows (²)
while in numerator linearly:
(k larger => f' smaller)

...that after couple of steps... $\ldots = k \cdot \left(1 - (tanh(k \cdot net))^2\right)$

Finally: $$f'_{bip}(k \cdot net) = k \cdot \left(1 - f^2_{bip}(k \cdot net)\right)$$

79

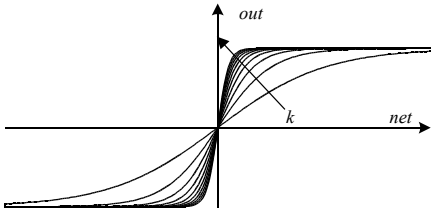
Soft activation functions



a)

- *S-curve, mapping net to values between (0, 1)*
- *0.5 centered*
- *Vanishing gradient!*

$$o_{uni} = f_{uni}(k \cdot net) = \frac{1}{1 + exp(-k \cdot net)}$$



b)

- *S-curve, mapping net to values between (-1, 1)*
- *tanh is zero-centered, capturing positive and negative net/out correlations*
- *Vanishing gradient!*

$$o_{bip} = f_{bip}(k \cdot net) = tanh(k \cdot net) = \frac{2}{1 + exp(-2 \cdot k \cdot net)} - 1$$

$$f'_{uni}(k \cdot net) = k \cdot o \cdot (1 - o)$$

$$f'_{bip}(k \cdot net) = k \cdot \left(1 - o^2_{bip}\right)$$

Unipolar
(or sigmoid w/ gain incorporated, 0 to 1)

Bipolar
(or hyperbolic tangent w/ gain incorp., 1- to +1)

80

Things to remember…

- **Gain…in soft act. function**
  - *A way to model "soft" threshold*
  - *Hard thresholds (hard act. fun.), may create a problem in real-world controls*
  - *Smaller gain is "safer" (continuous convergence), but learning is slower*
- **Alpha and gain**
  - *Two parameters at disposal to control process of training (learning)*
  - *Can be dynamically adjusted*
  - *Both can accelerate learning*
  - *But if increased too much, can lead to overshoot (alpha) or network saturation (gain)*
- **Selecting any dimensional space area**
  - *Just a generalization of "selecting area in 2D space" approach*
  - *Two layer network (neurons selecting an area, combined by AND neuron)*
  - *A simple form of "clustering"*
- **Gain…again**
  - *In a way defines a "range of active net" (when net changes, network output changes)*
  - *Reducing it may get network to "restart" learning again*
  - *Affects the value of first derivative*

Things to remember…

- **Training vs. testing dataset**
  - *Train on training dataset, test on testing dataset*
  - *Typically:*
    - *test data is NOT used for training*
    - *TE is calculated for training data (NOT for test data)*
- **Final solution**
  - *is a weight set (knowledge)*
  - *data driven (training data)*
  - *will depend on initial weight set*
  - *will depend on parameters chosen (network architecture, alpha, gain, etc.)*
- **How it relates to our projects:**
  - *In Project 1, there was NO training (you decided on decision line, i.e. you created neuron "by design")*
  - *In Project 2*
    - *Solution (weights) are obtained by the ALGORITHM (on TRAINING data)*
    - *TESTING data used for testing only (not training)*