BENG 420/520 Biomedical Data Analytics

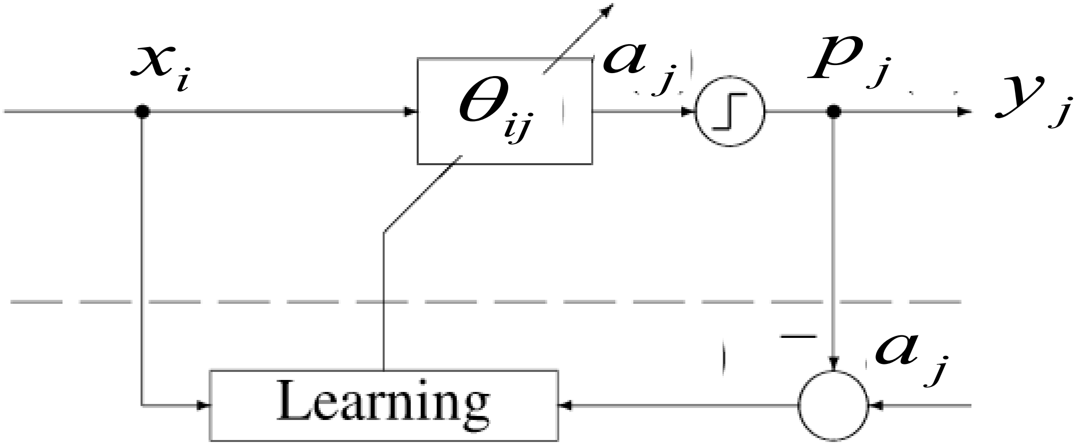
Prof. Qi Wei

Spring 2020

- Homework assignment #2 due on March 16, Monday
- Midterm on March 18, Wednesday
- Project report including introduction and methods due on March 23, Monday

Learning rules

- Idea: to adjust the weights of the network to minimize error on the training set



- Learning is formulated as an optimization search in weight space

NN Learning Algorithms (2)

- Delta rule (Least Mean Squared Error Rule, gradient descent) [Widrow and Hoff, 1960]

$$\Delta\theta_{ij} = \underset{\text{learning rate}}{\alpha}(\overset{\text{actual}}{y_j} - \overset{\text{predicted}}{t_j})g'(h_j)x_i$$

- Back propagation

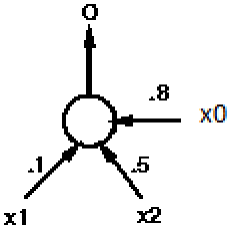
Perceptron Learning Algorithm (sequential)

```
Randomly initialize all weights $\theta_i$
Repeat until some stopping criterion is satisfied

    For one training example D^j=(X^j, y^j),j=1,2,…m
            compute neuron potential
                    $a = \sum_{i=0}^{n} \theta_i x_i$
            compute predicted output
                    $t^j = \begin{cases} 1, a \geq 0 \\ 0, a < 0 \end{cases}$
            if ( $y^j \neq t^j$ )
                    update every weight
                        $\Delta\theta_i = \alpha(y^j - t^j)x_i^j$
                        $\theta_i = \theta_i + \Delta\theta_i$
            end if
    end for
end repeat
```

Example: Learn OR in Perceptron

$$\Delta\Theta_i = \alpha(y_i - t_i)x_i$$



$\alpha = 0.2$

x0	x1	x2	y	a	t	$\Delta\theta_0$	θ_0	$\Delta\theta_1$	θ_1	$\Delta\theta_2$	θ_2
-1	-	-	-	-	-	-	0.8	-	0.1	-	0.5
-1 -0.8	0 0	0 0	0	-0.8	0	0	0.8	0	0.1	0	0.5
-1 -0.8	0 0	1 0.5	1	-0.3	0	-0.2	0.6	0	0.1	0.2	0.7
-1 -0.6	1 0.1	0 0	1	-0.5	0	-0.2	0.4	0.2	0.3	0	0.7
-1 -0.4	1 0.3	1 0.7	1	0.6	1	0	0.4	0	0.3	0	0.7
-1 -0.4	0 0	0 0	0	-0.4	0	0	0.4	0	0.3	0	0.7
-1 -0.4	0 0	1 0.7	1	0.3	1	0	0.4	0	0.3	0	0.7
-1 -0.4	1 0.3	0 0	1	-0.1	0	-0.2	0.2	0.2	0.5	0	0.7
-1 -0.2	1 0.5	1 0.7	1	1.0	1	0	0.2	0	0.5	0	0.7
-1 -0.2	0 0	0 0	0	-0.2	0	0	0.2	0	0.5	0	0.7
-1 -0.2	0 0	1 0.7	1	0.5	1	0		0		0	
-1 -0.2	1 0.5	0 0	1	0.3	1	0		0		0	
-1 -0.2	1 0.5	1 0.7	1	1.0	1	0		0		0	

Choosing learning rate

- If it is too small, it will take very long to converge
- If it is too big, weight updates may overshoot

- Optimal value is hard to determine and depend on the network.

- One can try different values (e.g., 1, 0.1, 0.01, 0.001)

Matlab Perceptron Example

- Initial guess
- Number of iterations
- Interpretation of weight vector?

- Example_myPerceptron.m

Batch vs sequential (online)

- Sequential (Online) training: update the weights immediately after processing each training pattern
  - Does not perform true gradient descent
  - Individual weight changes can be rather erratic
  - Overall learning can be quicker
- Batch training: update the weights after all training examples have been presented

Perceptron Learning Algorithm (sequential vs. batch)

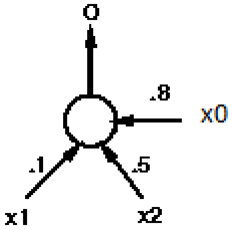
```
Randomly initialize all weights $\theta_i$
Repeat until some stopping criterion is satisfied
    For one training example $D^j=(X^j, y^j)$,$j=1,2,\ldots m$
        compute neuron potential
            $a = \sum_{i=0}^{n} \theta_i x_i$
        compute predicted output
            $t^j = \begin{cases} 1, a \geq 0 \\ 0, a < 0 \end{cases}$
        if ( $y^j \neq t^j$)
            update every weight
                $\Delta\theta_i = \alpha(y^j - t^j)x_i^j$
                $\theta_i = \theta_i + \Delta\theta_i$
        end if
    end for
end repeat
```

```
Randomly initialize weights $\theta_i$
Repeat until some stopping criterion is satisfied
    $\Delta\theta_i = 0;$
    For one training example $D^j=(X^j, y^j)$
        compute neuron potential
            $a = \sum_{i=0}^{n} \theta_i x_i$
        compute predicted output
            $t^j = \begin{cases} 1, a \geq 0 \\ 0, a < 0 \end{cases}$
        if ( $y^j \neq t^j$)
            store weight change
                $\Delta\theta_i = \Delta\theta_i + \alpha(y^j - t^j)x_i^j$
        end if
    end for
    update every weight
        $\theta_i = \theta_i + \frac{\Delta\theta_i}{m}$
end repeat
```

Sequential update vs. batch

- Matlab example_onePerceptron.m

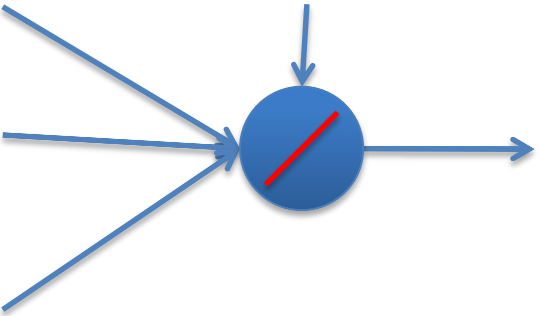
Example: Learn OR in Perceptron using batch



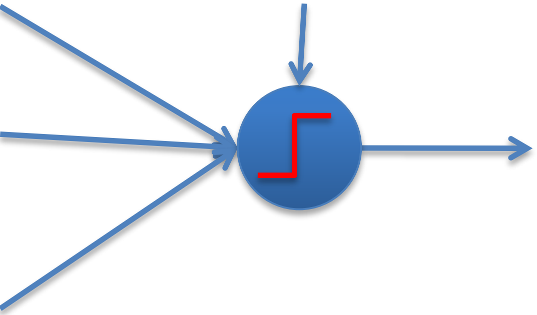
$\alpha = 0.2$

x0	x1	x2	y	t	$\Delta\theta_0$	θ_0	$\Delta\theta_1$	θ_1	$\Delta\theta_2$	θ_2
-1	-	-	-	-	-	0.8	-	0.1	-	0.5
-1	0	0	0							
-1	0	1	1							
-1	1	0	1							
-1	1	1	1							
-1	0	0	0							
-1	0	1	1							
-1	1	0	1							
-1	1	1	1							
-1	0	0	0							
-1	0	1	1							
-1	1	0	1							
-1	1	1	1							

Different "Neurons" – Activation functions



Linear Neuron



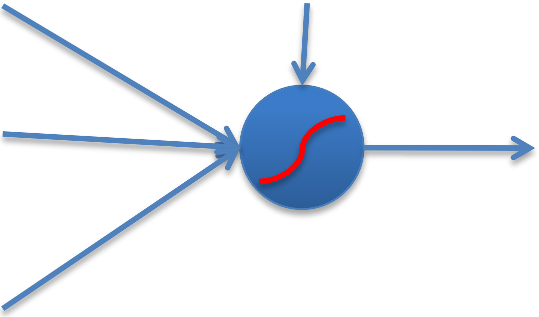
Perceptron

The perceptron is a linear classifier

Its "parameters" are a set of weights

We can learn the weights by adjusting the weights to decrease the error between the perceptron output and a set of known labels

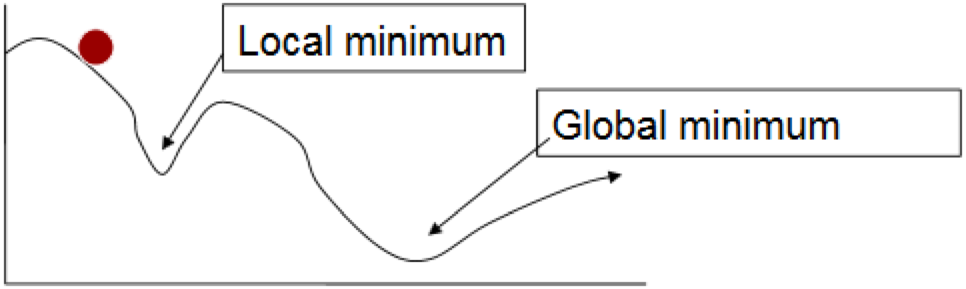
The weight adjustment algorithm is based on the principle of gradient descent



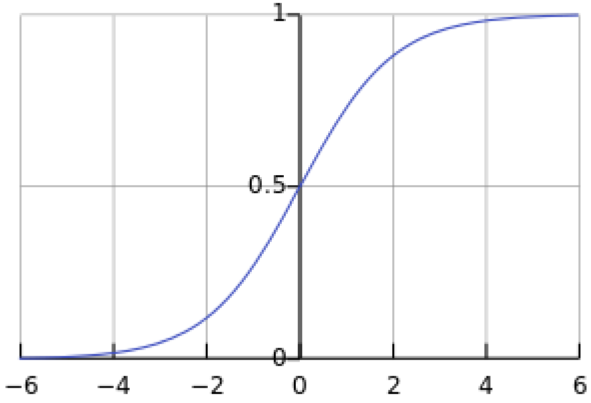
Logistic Neuron

Search Space

- ANN training is like marble rolling across a surface, which has valleys in it.
- Some valleys may be deeper than others – we want to find a deep one.
- When we find it, we stop training



Logistic Sigmoid Function



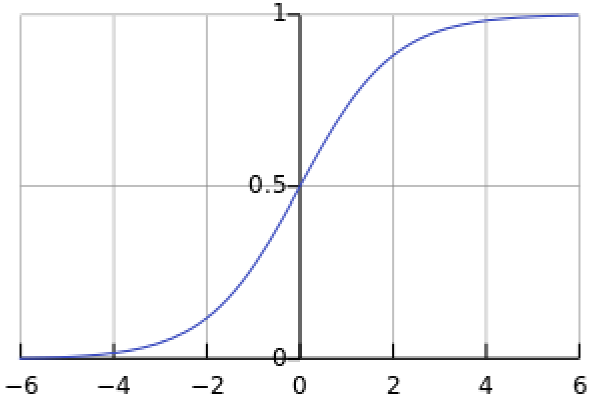
The activation of a sigmoid unit is a continuous function of its inputs that ranges between 0 and 1, increasing monotonically with its inputs.

Smooth nonlinearity!

$$t = f(a) = \frac{1}{1 + e^{-a}}$$

$$\frac{dt}{da}$$

Logistic Sigmoid Function



The activation of a sigmoid unit is a continuous function of its inputs that ranges between 0 and 1, increasing monotonically with its inputs.

Smooth nonlinearity!

$$t = f(a) = \frac{1}{1 + e^{-a}} \qquad \frac{dt}{da}$$

- Delta rule (Least Mean Squared Error Rule, gradient descent)

$$a = \sum_{i=0}^{n} \theta_i \cdot x_i^j = \theta \cdot X$$

$$\Delta\theta = \alpha(y^j - t^j) \cdot \frac{dt}{da} \cdot X^j$$

$$\theta \leftarrow \Delta\theta + \theta$$

Logistic neuron learning algorithm (sequential)

Input: a training set D={D^j=(X^j, y^j); j=1,2,…m}

Output: learned weights θ_i , i=0,1,…n;

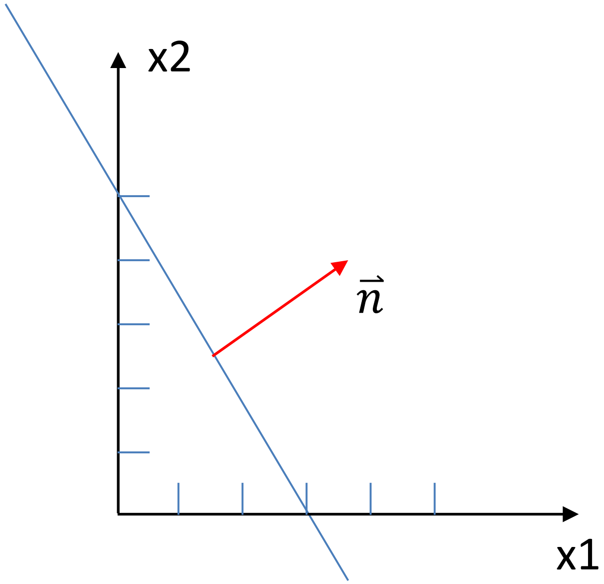
```
Randomly initialize all weights $\theta_i$
Repeat until some stopping criterion is satisfied
    For one training example D$^j$=($X^j$, $y^j$),j=1,2,…m;
        compute neuron potential:
            $a = \sum_{i=0}^{n} \theta_i X_i^j$
        compute predicted output:
            $t^j = \frac{1}{1+e^{-a}}$
        update every weight $\theta_i$ :
            $\Delta\theta_i = \alpha \cdot (y^j - t^j) \cdot \frac{dt}{da} \cdot X_i^j$
            $\theta_i = \theta_i + \Delta\theta_i$
        end if
    end for
end repeat
```

Biases versus Weights in Perceptrons

- Weights adjust slope of the decision boundary
- Bias adjusts zero crossing points

- Express $\vec{n}$ as a function of $\theta_1\ and\ \theta_2$

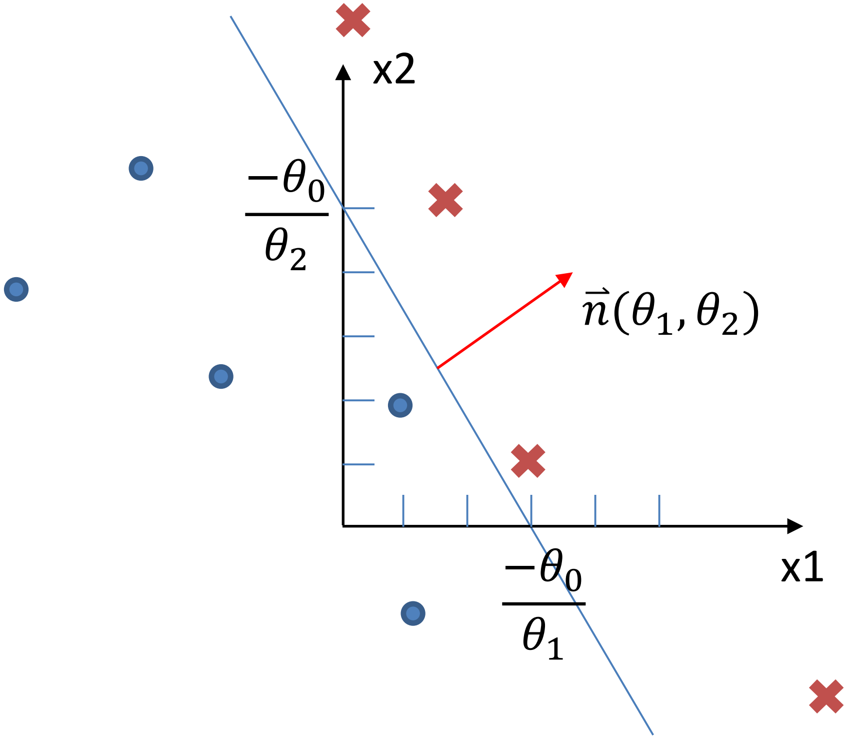
$$\theta_1 \cdot x_1 + \theta_2 \cdot x_2 + \theta_0 = 0$$



Perceptron Decision Boundary

Red points belong to the positive class, blue points belong to the negative class

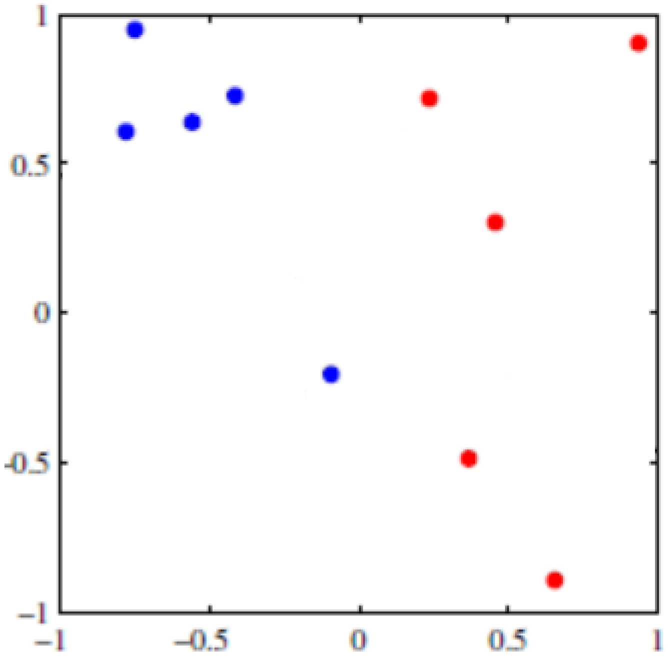
When an error is made, moves the weight in a direction that corrects the error


x2
$\frac{-\theta_0}{\theta_2}$
$\vec{n}(\theta_1, \theta_2)$
$\frac{-\theta_0}{\theta_1}$
x1

Perceptron decision boundary

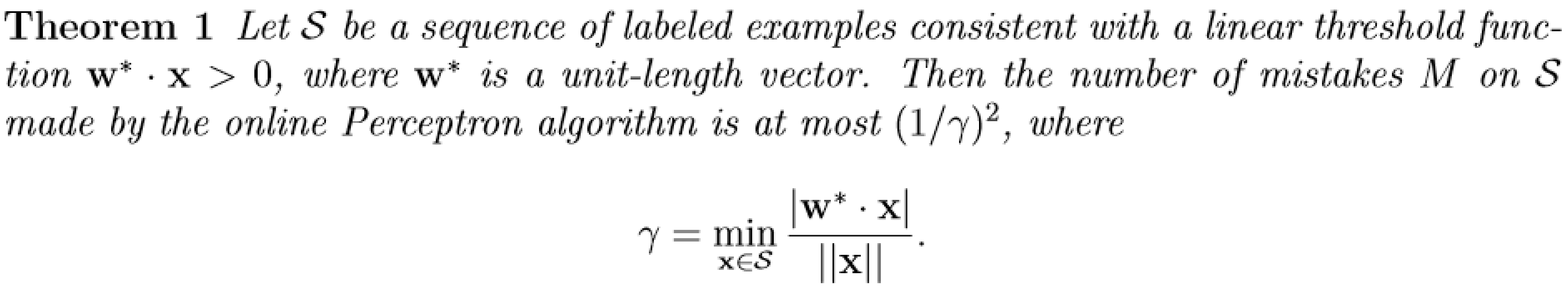
Red points belong to the positive class, blue points belong to the negative class

When an error is made, moves the weight in a direction that corrects the error



Perceptron Convergence Theorem

- If a set of examples are learnable (i.e., 100% correct classification), the Perceptron Learning Rule will find the necessary weights (Minksy and Papert, 1988) In a finite number of steps



Theorem 1 *Let $\mathcal{S}$ be a sequence of labeled examples consistent with a linear threshold function $\mathbf{w}^* \cdot \mathbf{x} > 0$, where $\mathbf{w}^*$ is a unit-length vector. Then the number of mistakes M on $\mathcal{S}$ made by the online Perceptron algorithm is at most $(1/\gamma)^2$, where*

$$\gamma = \min_{\mathbf{x} \in \mathcal{S}} \frac{|\mathbf{w}^* \cdot \mathbf{x}|}{||\mathbf{x}||}.$$

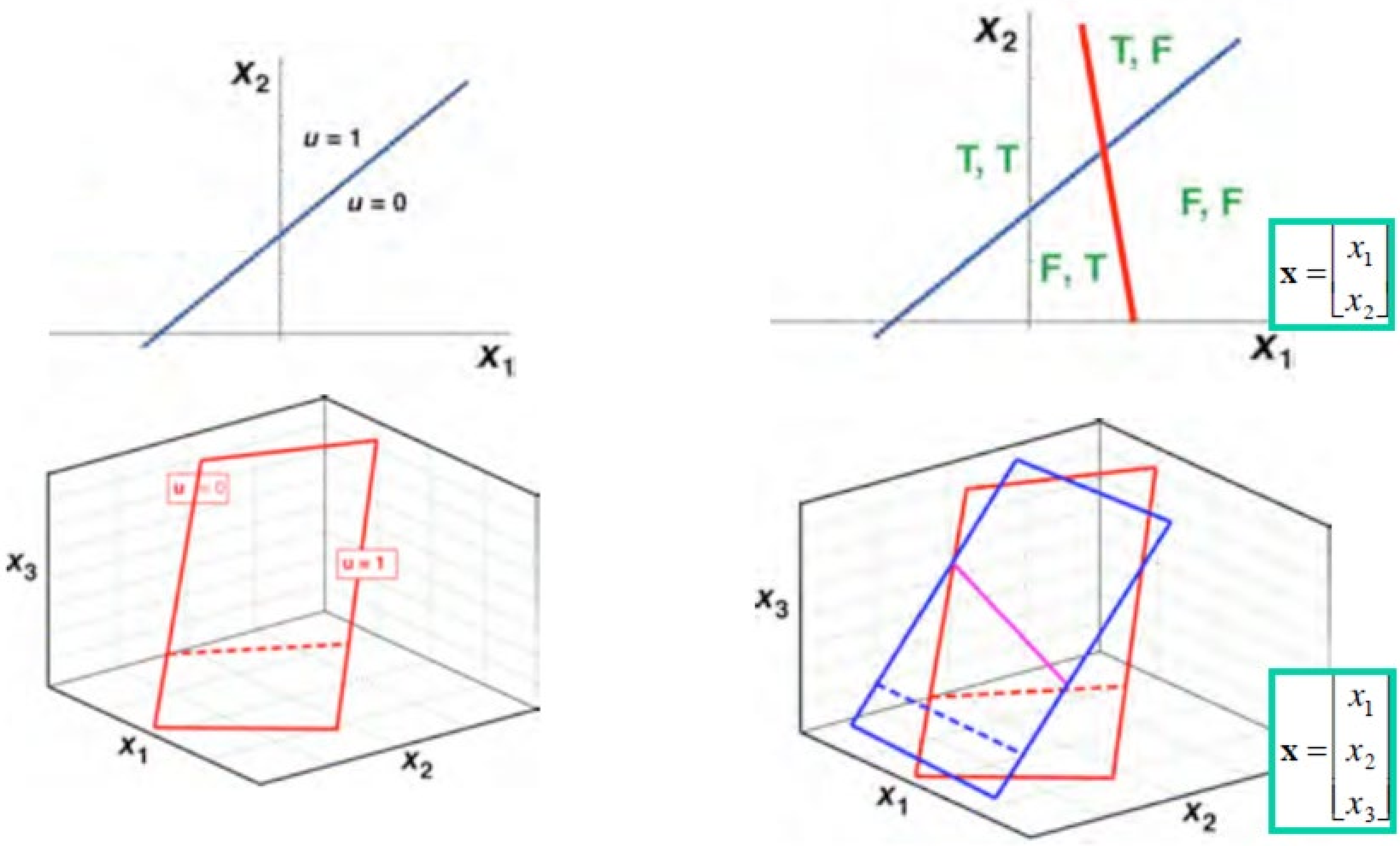
•If we scale all examples to have Euclidean length 1, then r is the min distance of any example to the plane w* . X = 0

• r is called the margin of w*

• r without absolute is the cosine of the angle between w* and x

Single-layer, single-node or multi-node

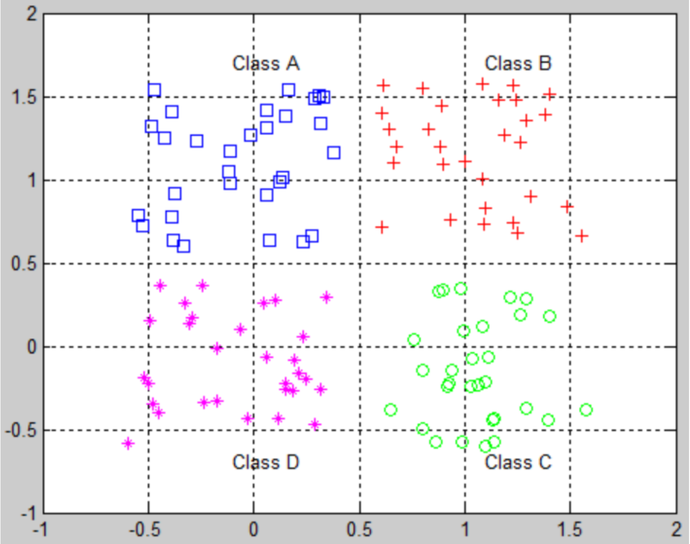
- More outputs lead to more discriminants (classes)



x_2
$u = 1$
$u = 0$
x_1
$u = 0$
$u = 1$
x_3
x_1
x_2
x_2
T, F
T, T
F, F
F, T
x_1
$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$
x_3
x_1
x_2
$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$

Perceptron Example

- Four class classification



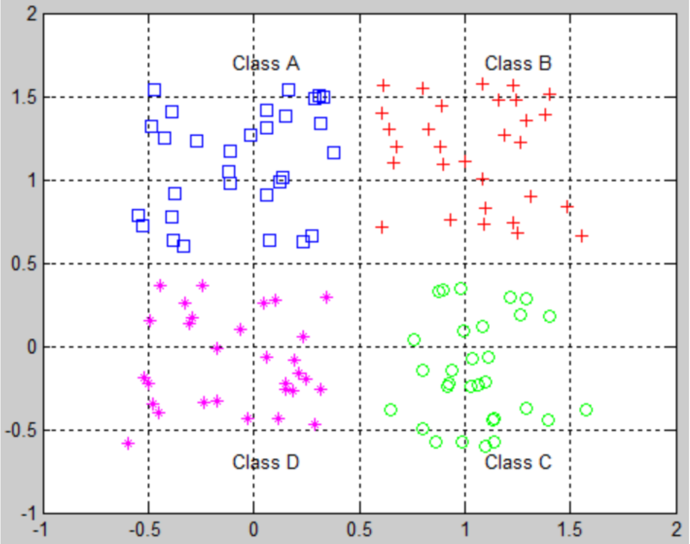
- Which label coding would work? If not, give feasible labels
  - A[0 1]; B[1 0]; C[0 0]; D[1 1]
  - A[1 1]; B[1 0]; C[1 1]; D[0 0]
  - A[0 0]; B[1 1]; C[0 0]; D[1 1]
  - A[0 1]; B[1 1]; C[1 0]; D[0 0]
  - A[1 1]; B[1 0]; C[1 1]; D[1 0]

example_twoPerceptrons.m

Perceptron Example

- Four class classification

  example_twoPerceptrons.m

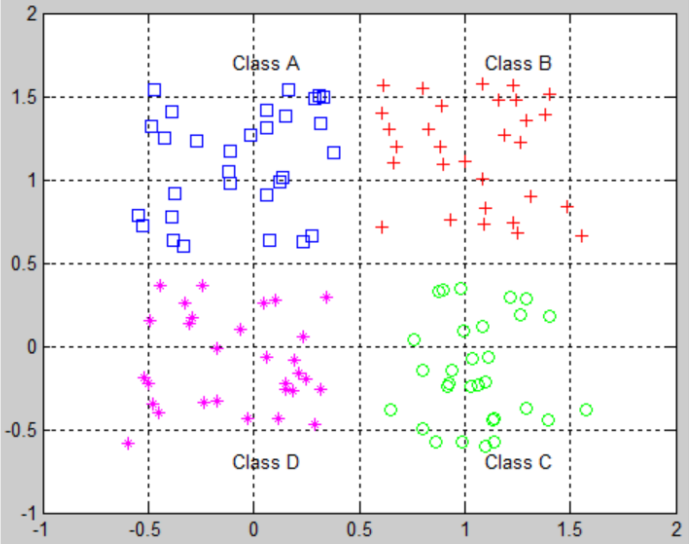


- Where would the decision boundaries be?

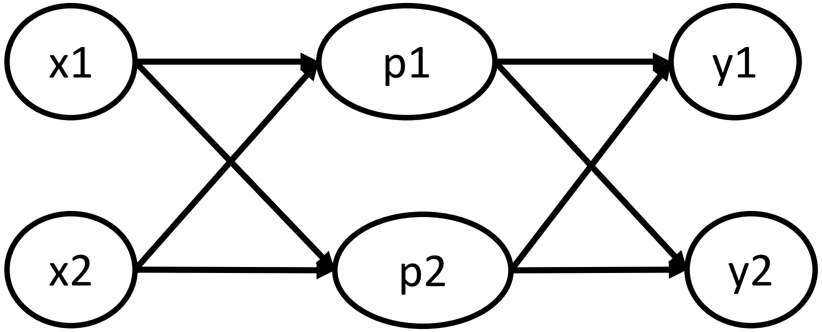
Perceptron Example

- Four class classification

  example_twoPerceptrons.m

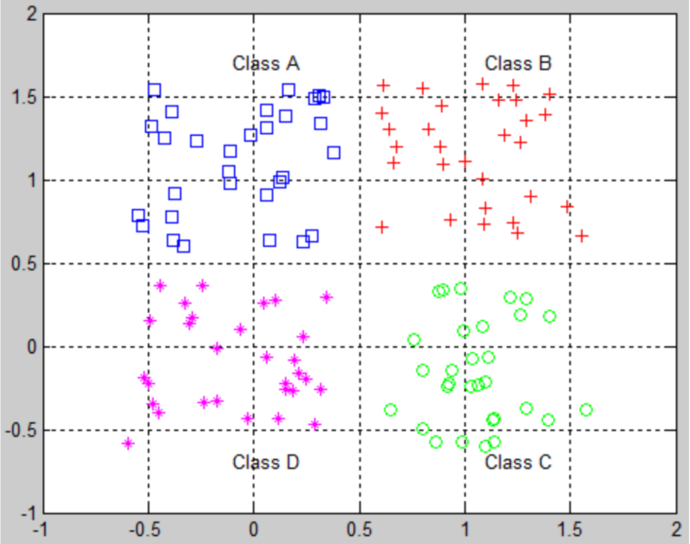


- What would be the network architecture?



Perceptron Example

- Four class classification



- Which label coding would work? If not, give feasible labels
  - A[0 1]; B[1 0]; C[0 0]; D[1 1]
  - A[1 1]; B[1 0]; C[1 1]; D[0 0]
  - A[0 0]; B[1 1]; C[0 0]; D[1 1]
  - A[0 1]; B[1 1]; C[1 0]; D[0 0]
  - A[1 1]; B[1 0]; C[1 1]; D[1 0]

example_twoPerceptrons.m

Acknowledgement

- http://www.cs.cornell.edu/courses/cs4700