CS4442/9542b Artificial Intelligence II

prof. Olga Veksler

Lecture 5

Machine Learning

Linear Classifier

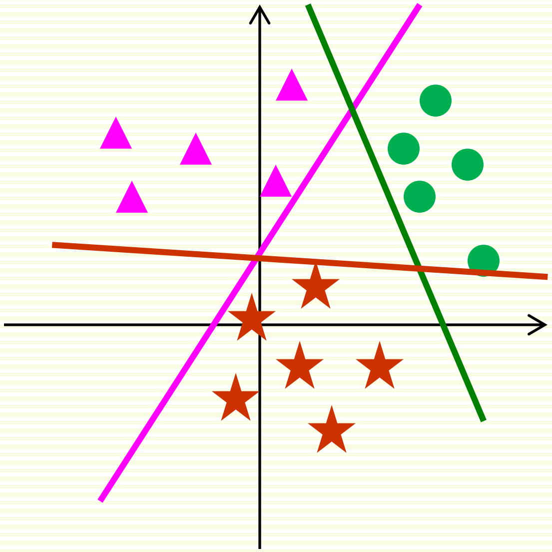
Multiple Classes

Outline

- Linear Classifier
  - Multiple classes
    1. Use collection of 2-class classifiers
       - one vs. all
       - all pairs
    2. Design multi-class loss functions
       - Perceptron Loss Function
       - Softmax Loss Function
  - Weight Regularization

Using 2-class Case: One vs. All

- Have classes 1, 2, … , **m**
- Can construct multi-class classifier based on 2-class classifiers
- One way
  - Assume each 2-class classifier also gives confidence
    - Distance from separating hyperplane
      - Higher distance, more confidence
  - Train **m** 2-class classifiers
    - **1** vs other classes
    - **2** vs. other classes
    - ….
    - **m** vs. other classes
    - Make sure number of examples is balanced during training



  - At test time, run new sample through **m** binary classifiers
    - highest confidence class “wins”
- Works for any type of 2-class classifier, not just linear

Using 2-class Case: All pairs

- Train 2-class classifier for each distinct pair of classes (**i,j**)

	1	2	3	4
1		✓	✓	✓
2			✓	✓
3				✓
4				

- At test time, run new example **x** through all binary classifiers
  - Choose most frequently occurring class
  - For example, **x** was classified
    - 1 time as class 1
    - 2 times as class 2
    - 0 times as class 3
    - 3 times as class 4

decide class 4

Multiple Classes: General Case

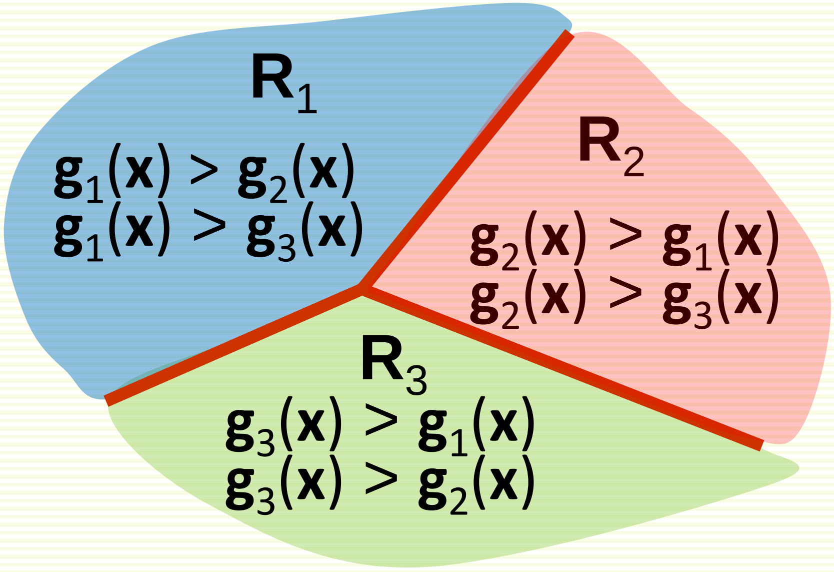
- General multiclass case
  - not based on 2-class classifiers
- Define **m** linear discriminant functions

$$\mathbf{g}_i(\mathbf{x}) = \mathbf{w}_i^t\mathbf{x} + \mathbf{w}_{i0} \text{ for } \mathbf{i} = 1, 2, \ldots \mathbf{m}$$

- Assign **x** to class **i** if

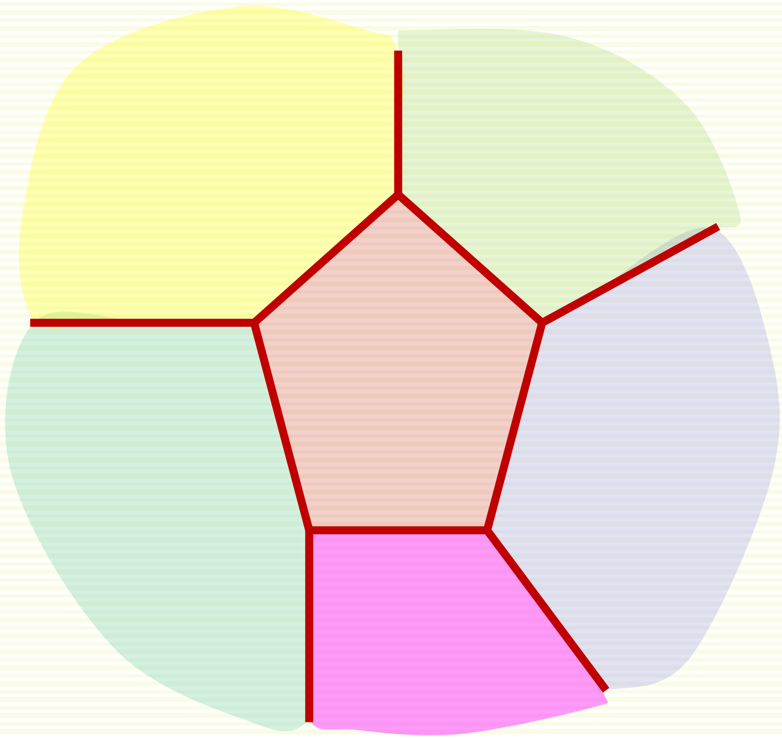
$$\mathbf{g}_i(\mathbf{x}) > \mathbf{g}_j(\mathbf{x}) \text{ for all } \mathbf{j} \neq \mathbf{i}$$

- Let $\mathbf{R}_i$ be decision region for class **i**
  - all samples in $\mathbf{R}_i$ assigned to class **i**



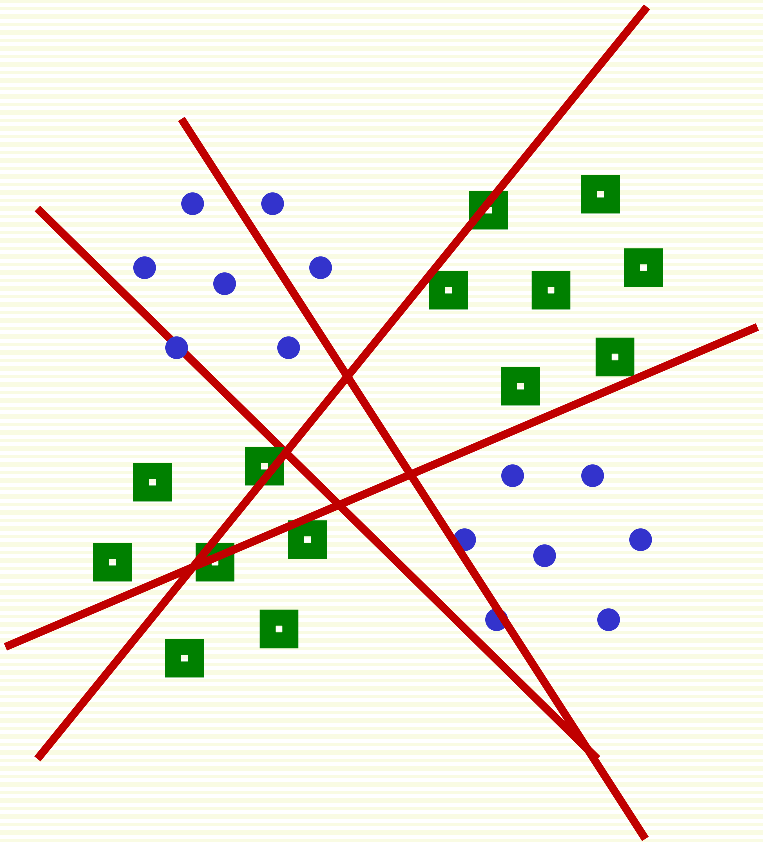
Multiple Classes

- Can be shown that decision regions are convex
- In particular, they must be spatially contiguous



Failure Case for Linear Classifier

- Thus applicability of linear classifiers is limited to mostly unimodal distributions, such as Gaussian
- For not unimodal data, need non-contiguous decision regions
- Linear classifier will fail

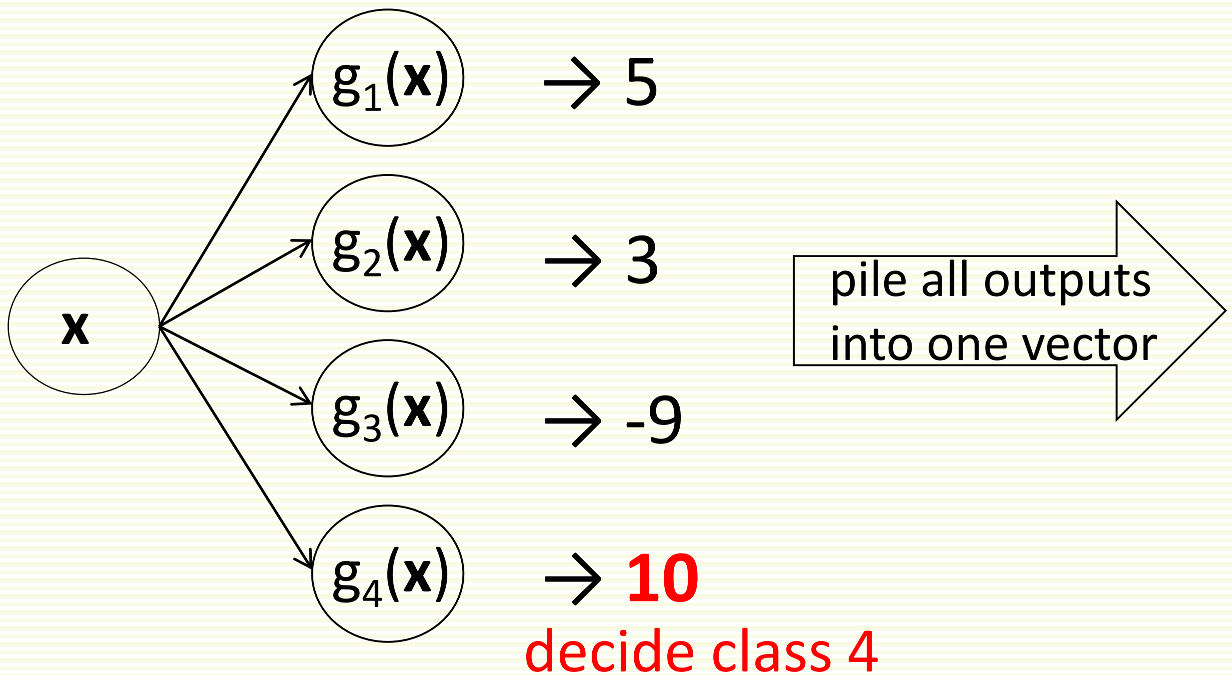


Multiclass Linear Classifier: Matrix Notation

- Assume examples **x** are augmented with extra feature 1, no need to write bias explicitly
  - but from now on will not change notation to **z**'s
- Define **m** discriminant functions

$$\mathbf{g_i}(\mathbf{x}) = \mathbf{w}_i^t\mathbf{x} \quad \text{for } \mathbf{i} = 1, 2, \ldots \mathbf{m}$$

- Assign **x** to **i** that gives maximum $\mathbf{g_i}(\mathbf{x})$
- Picture illustration



$$\begin{bmatrix} 5 \\ 3 \\ -9 \\ 10 \end{bmatrix}$$

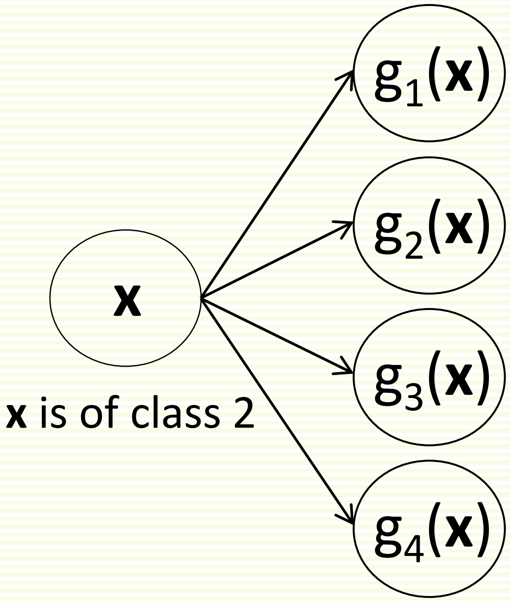
Multiclass Linear Classifier: Matrix Notation

- Could use one dimensional output $y_i \in \{1, 2, 3, \ldots, m\}$
- Convenient to use multi-dimensional outputs

$$y^j = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \qquad y^j = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \qquad y^j = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} \qquad y^j = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

class 1 **class 2** **class 3** **class 4**

- For training, if sample is of class **i**, want output vector to be 0 everywhere except position **i**, where it should be 1

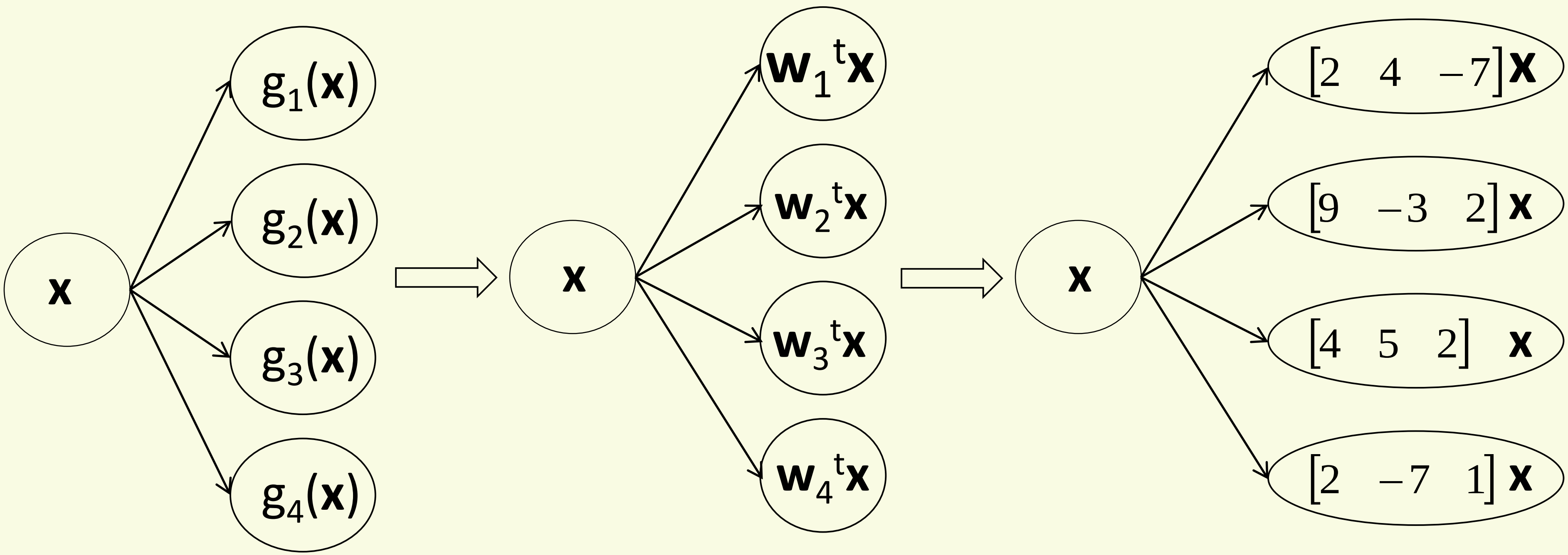


$$\begin{bmatrix} 5 \\ 3 \\ -9 \\ 10 \end{bmatrix} \qquad \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$

got this want this

Multiclass Linear Classifier: Matrix Notation

- Assign $\mathbf{x}$ to $\mathbf{i}$ that gives maximum $\mathbf{g_i}(\mathbf{x}) = \mathbf{w}_i^t\mathbf{x}$



- In matrix notation

$$\begin{array}{c} \mathbf{w}_1 \\ \mathbf{w}_2 \\ \mathbf{w}_3 \\ \mathbf{w}_4 \end{array}\underbrace{\begin{bmatrix} 2 & 4 & -7 \\ 9 & -3 & 2 \\ 4 & 5 & 2 \\ 2 & -7 & 1 \end{bmatrix}}_{\mathbf{W}} \underbrace{\begin{bmatrix} 1 \\ 7 \\ 4 \end{bmatrix}}_{\mathbf{x}} = \underbrace{\begin{bmatrix} 2 \\ -4 \\ \boxed{47} \\ -43 \end{bmatrix}}_{\mathbf{Wx}}$$

- Assign $\mathbf{x}$ to class that corresponds to largest row of $\mathbf{Wx}$

Quadratic Loss Function

- Assign sample $\mathbf{x}^i$ to class that corresponds to largest row of $\mathbf{W}\mathbf{x}^i$
- Loss function?

$$\underset{\mathbf{W}\mathbf{x}^i}{\begin{bmatrix} 2 \\ -4 \\ 47 \\ -43 \end{bmatrix}} \qquad \underset{\mathbf{y}^i}{\begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}}$$

- Can use quadratic loss per sample $\mathbf{x}^i$ as $½||\mathbf{W}\mathbf{x}^i - \mathbf{y}^i||^2$
  - for example above, loss $(2^2 + 4^2 + 47^2 + 44^2)/2$
  - total loss on all training samples $L(\mathbf{W}) = ½\Sigma_i|| \mathbf{W}\mathbf{x}^i - \mathbf{y}^i ||^2$
  - gradient of the loss

$$\nabla \mathbf{L}(\mathbf{W}) = \sum_{\mathbf{i}} \left(\mathbf{W}\mathbf{x}^{\mathbf{i}} - \mathbf{y}^{\mathbf{i}}\right)\left(\mathbf{x}^{\mathbf{i}}\right)^{\mathbf{t}}$$

  - ∇ **L**(**W**) has the same shape as the same shape as **W**
  - batch gradient descent update

$$\mathbf{W} = \mathbf{W} - \alpha \sum_{\mathbf{i}} \left(\mathbf{W}\mathbf{x}^{\mathbf{i}} - \mathbf{y}^{\mathbf{i}}\right)\left(\mathbf{x}^{\mathbf{i}}\right)^{\mathbf{t}}$$

Quadratic Loss Function

- Consider gradient descent update, single sample **x** with $\alpha = 1$

$$\mathbf{W} = \mathbf{W} - (\mathbf{W}\mathbf{x} - \mathbf{y})\mathbf{x}^t$$

- Suppose $\mathbf{x} = \begin{bmatrix} 1 \\ 3 \\ 2 \end{bmatrix}$ and is of class 2 and $\mathbf{W} = \begin{bmatrix} 2 & 4 & -7 \\ 9 & -3 & 2 \\ 4 & 5 & 2 \\ 2 & -7 & 1 \end{bmatrix}$

ok

too large

too small

$$\mathbf{W}\mathbf{x} - \mathbf{y} = \begin{bmatrix} 0 \\ 4 \\ 23 \\ -17 \end{bmatrix} - \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 3 \\ 23 \\ -17 \end{bmatrix}$$

- update rule

$$\mathbf{W} = \begin{bmatrix} 2 & 4 & -7 \\ 9 & -3 & 2 \\ 4 & 5 & 2 \\ 2 & -7 & 1 \end{bmatrix} - \begin{bmatrix} 0 \\ 3 \\ 23 \\ -17 \end{bmatrix} \begin{bmatrix} 1 & 3 & 2 \end{bmatrix} = \begin{bmatrix} 2 & 4 & -7 \\ 9 & -3 & 2 \\ 4 & 5 & 2 \\ 2 & -7 & 1 \end{bmatrix} - \begin{bmatrix} 0 & 0 & 0 \\ 3 & 9 & 6 \\ 23 & 69 & 46 \\ -17 & -51 & -34 \end{bmatrix} = \begin{bmatrix} 2 & 4 & -7 \\ 6 & -12 & -4 \\ -19 & -64 & -44 \\ 19 & 44 & 35 \end{bmatrix}$$

Quadratic Loss Function

ok
too large
too small

$$\mathbf{Wx} - \mathbf{y} = \begin{bmatrix} 0 \\ 4 \\ 23 \\ -17 \end{bmatrix} - \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 3 \\ 23 \\ -17 \end{bmatrix}$$

- With new $\mathbf{W} = \begin{bmatrix} 2 & 4 & -7 \\ 6 & -12 & -4 \\ -19 & -64 & -44 \\ 19 & 44 & 35 \end{bmatrix}$, $\quad \mathbf{Wx} = \begin{bmatrix} 0 \\ -38 \\ -299 \\ 221 \end{bmatrix}$

- Already saw that quadratic loss does not work that well for classification

Perceptron Loss

- Generalize Perceptron loss to multiclass setting
- Per-example loss: largest score minus score for the correct class

$$\underbrace{\begin{bmatrix} 2 \\ -4 \\ 47 \\ -43 \end{bmatrix}}_{\mathbf{Wx}^i} \qquad \underbrace{\begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}}_{\mathbf{y}^i}$$

Loss is 47-(-43)= 90

$$\underbrace{\begin{bmatrix} 20 \\ 40 \\ 17 \\ -43 \end{bmatrix}}_{\mathbf{Wx}^i} \qquad \underbrace{\begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}}_{\mathbf{y}^i}$$

Loss is 40-40= 0

- Formula for Perceptron loss on sample $\mathbf{x}^i$

$$\mathbf{L}_i(\mathbf{W}) = \max_{\mathbf{k}}[(\mathbf{Wx}^i)_k - (\mathbf{Wx}^i)_{\mathbf{c}}]$$

  - $(\mathbf{Wx}^i)_{\mathbf{k}}$ is the entry in row **k** of vector $\mathbf{Wx}^i$
  - **c** is the correct class of sample $\mathbf{x}^i$

Perceptron Loss Function

- Gradient of loss on one example
  - **c** is the correct class row
  - **r** is the row where $\mathbf{W}\mathbf{x}^i$ is largest
  - if **r = c**, $-\nabla \mathbf{L_i}(\mathbf{W}) = 0$
  - otherwise, $-\nabla \mathbf{L_i}(\mathbf{W}) = \begin{array}{c} \left[\begin{array}{cccc} 0 & 0 & 0 & 0 \\ & -\mathbf{x}^i & & \\ 0 & 0 & 0 & 0 \\ & \mathbf{x}^i & & \end{array}\right] \end{array}\begin{array}{l} \\ \text{row } \mathbf{r} \\ \\ \text{row } \mathbf{c} \end{array}$
- Example

$$\underset{\mathbf{W}\mathbf{x}^i}{\begin{bmatrix} 2 \\ -4 \\ 47 \\ -43 \end{bmatrix}} \quad \underset{\mathbf{x}^i}{\begin{bmatrix} 1 \\ 3 \\ 2 \end{bmatrix}} \quad \underset{\mathbf{y}^i}{\begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}}$$

$$-\nabla \mathbf{L_i}(\mathbf{W}) = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ -1 & -3 & -2 \\ 1 & 3 & 2 \end{bmatrix}$$

Perceptron Loss Function: Example Cont.

$$-\nabla \mathbf{L_i}(\mathbf{W}) = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ -1 & -3 & -2 \\ 1 & 3 & 2 \end{bmatrix} \qquad \mathbf{x}^i = \begin{bmatrix} 1 \\ 3 \\ 2 \end{bmatrix} \qquad \underset{\mathbf{y}^i}{\begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}}$$

- With α = 1, new $\mathbf{W} = \begin{bmatrix} 2 & 4 & -7 \\ 9 & -3 & 2 \\ 4 & 5 & 2 \\ 2 & -7 & 1 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ -1 & -3 & -2 \\ 1 & 3 & 2 \end{bmatrix} = \begin{bmatrix} 2 & 4 & -7 \\ 9 & -3 & 2 \\ 3 & 2 & 0 \\ 3 & -4 & 3 \end{bmatrix}$

- With new weights

$$\mathbf{W}\mathbf{x}^i = \begin{bmatrix} 0 \\ 4 \\ 9 \\ -3 \end{bmatrix}$$

- Compare to the old weights

$$\mathbf{W}_{old}\mathbf{x}^i = \begin{bmatrix} 0 \\ 4 \\ 23 \\ -17 \end{bmatrix}$$

ok

too large

too small

Softmax Function

- Define softmax(**a**) function

$$\begin{bmatrix} \mathbf{a}_1 \\ \mathbf{a}_2 \\ \mathbf{a}_3 \\ \mathbf{a}_4 \end{bmatrix} \xrightarrow{\text{softmax}} \begin{bmatrix} \dfrac{\exp(\mathbf{a}_1)}{\sum_{j=1}^{4} \mathbf{exp}(\mathbf{a}_j)} \\ \dfrac{\exp(\mathbf{a}_2)}{\sum_{j=1}^{4} \mathbf{exp}(\mathbf{a}_j)} \\ \dfrac{\exp(\mathbf{a}_3)}{\sum_{j=1}^{4} \mathbf{exp}(\mathbf{a}_j)} \\ \dfrac{\exp(\mathbf{a}_4)}{\sum_{j=1}^{4} \mathbf{exp}(\mathbf{a}_j)} \end{bmatrix}$$

- Example

$$\begin{bmatrix} -3 \\ 2 \\ 1 \end{bmatrix} \xrightarrow{\text{softmax}} \begin{bmatrix} \dfrac{\exp(-3)}{\mathbf{exp}(-3)+\mathbf{exp}(2)+\mathbf{exp}(1)} \\ \dfrac{\exp(2)}{\mathbf{exp}(-3)+\mathbf{exp}(2)+\mathbf{exp}(1)} \\ \dfrac{\exp(1)}{\mathbf{exp}(-3)+\mathbf{exp}(2)+\mathbf{exp}(1)} \end{bmatrix}$$

$$= \begin{bmatrix} 0.005 \\ 0.7275 \\ 0.2676 \end{bmatrix}$$

- Softmax renormalizes a vector so that it can be interpreted as a vector of probabilities

Softmax Loss Function

- Generalization of logistic regression to multiclass case

- Instead of raw scores $\begin{bmatrix} w_1^T x \\ w_2^T x \\ w_3^T x \\ w_4^T x \end{bmatrix} = \begin{bmatrix} 2 \\ -1 \\ 5 \\ -3 \end{bmatrix}$

- Use softmax scores

$$\textbf{softmax}\left(\begin{bmatrix} w_1^T x \\ w_2^T x \\ w_3^T x \\ w_4^T x \end{bmatrix}\right) = \textbf{softmax}\left(\begin{bmatrix} 2 \\ -1 \\ 5 \\ -3 \end{bmatrix}\right) = \begin{bmatrix} 0.0473 \\ 0.0024 \\ 0.9500 \\ 0.0003 \end{bmatrix} = \begin{bmatrix} \mathbf{Pr}(\mathbf{class}1) \\ \mathbf{Pr}(\mathbf{class}2) \\ \mathbf{Pr}(\mathbf{class}3) \\ \mathbf{Pr}(\mathbf{class}4) \end{bmatrix}$$

- Classifier output interpreted as probability for each class

Gradient Descent: Softmax Loss Function

- Optimize under $-\log \Pr(\mathbf{y}^i)$ loss function
- Example

$$\mathbf{x}^i = \begin{bmatrix} 1 \\ 3 \\ 2 \end{bmatrix} \quad \mathbf{y}^i = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \qquad \mathbf{W} = \begin{bmatrix} 2 & 4 & -7 \\ 9 & -3 & 2 \\ 4 & 5 & 2 \\ 2 & -7 & 1 \end{bmatrix}$$

$$\mathbf{softmax}(\mathbf{Wx}^i) = \mathbf{softmax}\left(\begin{bmatrix} 0 \\ 4 \\ 23 \\ -17 \end{bmatrix}\right) = \begin{bmatrix} 0.000000000102619 \\ 0.000000005602796 \\ 0.999999994294585 \\ 0.000000000000001 \end{bmatrix} = \begin{bmatrix} \mathbf{Pr}(\mathbf{class}1) \\ \mathbf{Pr}(\mathbf{class}2) \\ \mathbf{Pr}(\mathbf{class}3) \\ \mathbf{Pr}(\mathbf{class}4) \end{bmatrix}$$

- Loss on this example is $-\log(0.000000000000001) = 40$

Gradient Descent: Softmax Loss Function

- Update rule for weight matrix **W**

$$W = W + \alpha \sum_{i} \left(y^{i} - \operatorname{softmax}\left(Wx^{i}\right)\right)\left(x^{i}\right)^{t}$$

- Example, single sample gradient descent with α = 0.1

$$\mathbf{x}^{i} = \begin{bmatrix} 1 \\ 3 \\ 2 \end{bmatrix} \qquad \underset{\mathbf{y}^{i}}{\begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}} \qquad \mathbf{W} = \begin{bmatrix} 2 & 4 & -7 \\ 9 & -3 & 2 \\ 4 & 5 & 2 \\ 2 & -7 & 1 \end{bmatrix} \qquad \mathbf{Wx}^{i} = \begin{bmatrix} 0 \\ 4 \\ 23 \\ -17 \end{bmatrix}$$

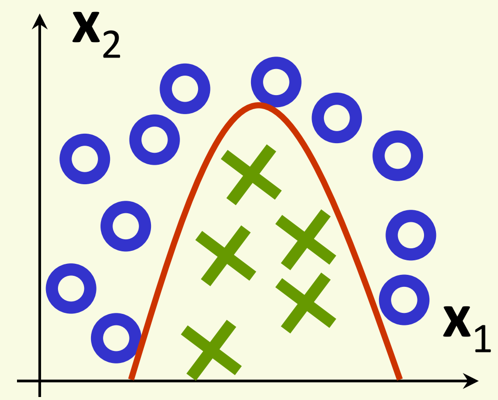
- Update for **W**

$$\mathbf{W} = \begin{bmatrix} 2 & 4 & -7 \\ 9 & -3 & 2 \\ 4 & 5 & 2 \\ 2 & -7 & 1 \end{bmatrix} + 0.1\left(\begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} - \mathbf{softmax}\begin{pmatrix} 0 \\ 4 \\ 23 \\ -17 \end{pmatrix}\right)\begin{bmatrix} 1 & 3 & 2 \end{bmatrix} = \begin{bmatrix} 2 & 4 & -7 \\ 9 & -3 & 2 \\ 3.9 & 4.7 & 1.8 \\ 2.1 & -6.7 & 1.2 \end{bmatrix}$$

Generalized Linear Classifier

- Can use other discriminant functions, like quadratics

  $\mathbf{g}(\mathbf{x}) = \mathbf{w}_0+\mathbf{w}_1\mathbf{x}_1+\mathbf{w}_2\mathbf{x}_2+ \mathbf{w}_{12}\mathbf{x}_1\mathbf{x}_2 +\mathbf{w}_{11}\mathbf{x}_1^{\,2} +\mathbf{w}_{22}\mathbf{x}_2^{\,2}$



- Methodology is almost the same as in the linear case
  - $\mathbf{f}(\mathbf{x}) = \text{sign}(\mathbf{w}_0+\mathbf{w}_1\mathbf{x}_1+\mathbf{w}_2\mathbf{x}_2+\mathbf{w}_{12}\mathbf{x}_1\mathbf{x}_2 +\mathbf{w}_{11}\mathbf{x}_1^{\,2} + \mathbf{w}_{22}\mathbf{x}_2^{\,2})$
  - $\mathbf{z} = [\,1 \quad \mathbf{x}_1 \quad \mathbf{x}_2 \quad \mathbf{x}_1\mathbf{x}_2 \quad \mathbf{x}_1^{\,2} \quad \mathbf{x}_2^{\,2}]$
  - $\mathbf{a} = [\,\mathbf{w}_0 \quad \mathbf{w}_1 \quad \mathbf{w}_2 \quad \mathbf{w}_{12} \quad \mathbf{w}_{11} \quad \mathbf{w}_{22}]$
  - use gradient descent to minimize Perceptron loss function, any other loss function
- Can add any degree polynomial features

Generalized Linear Classifier

- Generalized linear classifier

$$\mathbf{g}(\mathbf{x},\mathbf{w}) = \mathbf{w}_0+\sum_{i=1\ldots m} \mathbf{w}_i\mathbf{h}_i(\mathbf{x})$$

- $\mathbf{h}(\mathbf{x})$ are called basis function, can be arbitrary functions
  - in strictly linear case, $\mathbf{h}_i(\mathbf{x}) = \mathbf{x}_i$

- Linear function in its parameters **w**

$$\mathbf{g}(\mathbf{x},\mathbf{w}) = \mathbf{w}_0+\mathbf{w}^t\mathbf{h}$$

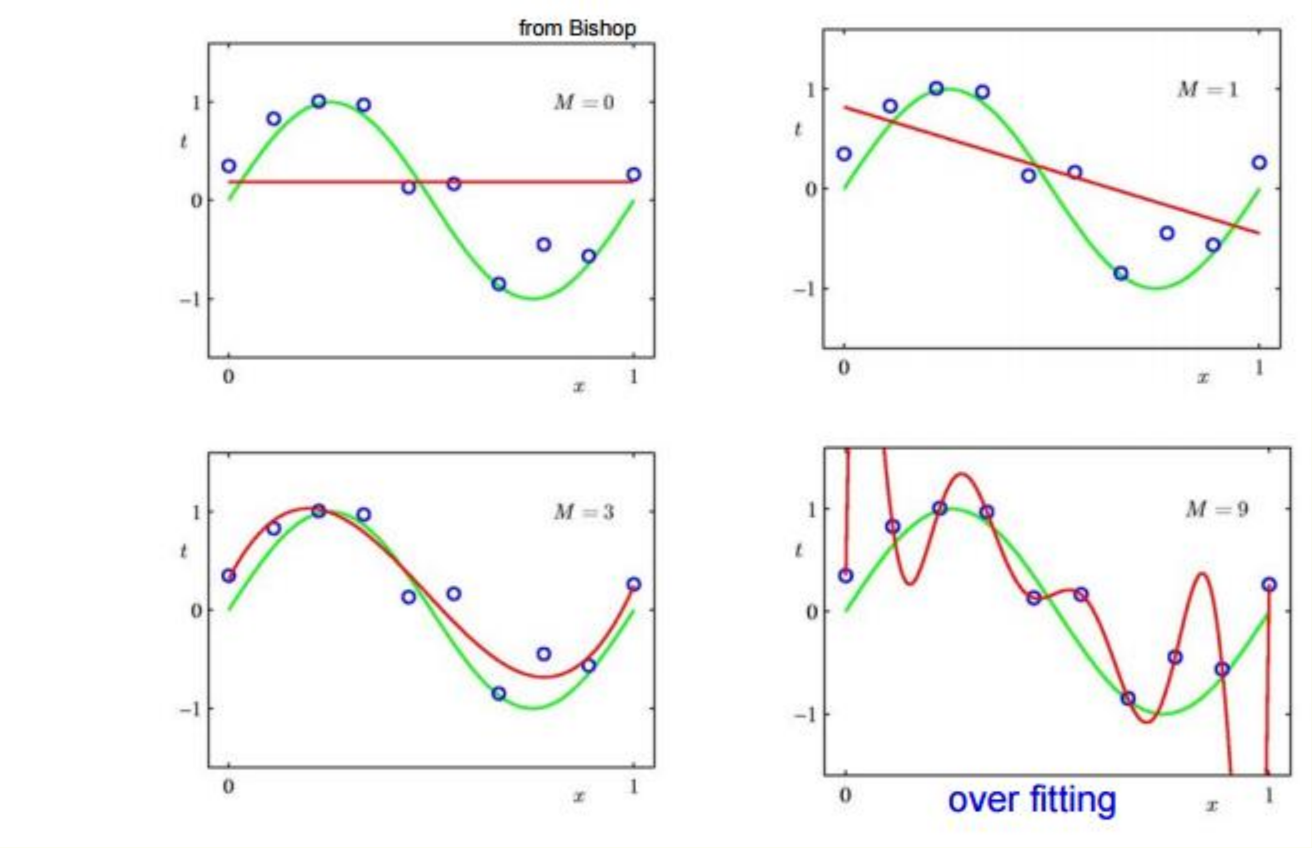
$$\mathbf{h} = [\mathbf{h}_1(\mathbf{x}) \;\; \mathbf{h}_2(\mathbf{x}) \;\; \ldots \;\; \mathbf{h}_m(\mathbf{x})]$$

$$[\mathbf{w}_1 \quad\;\; \mathbf{w}_2 \quad\; \ldots \quad\; \mathbf{w}_m]$$

- Use the same training methods as before with new feature vector **h**

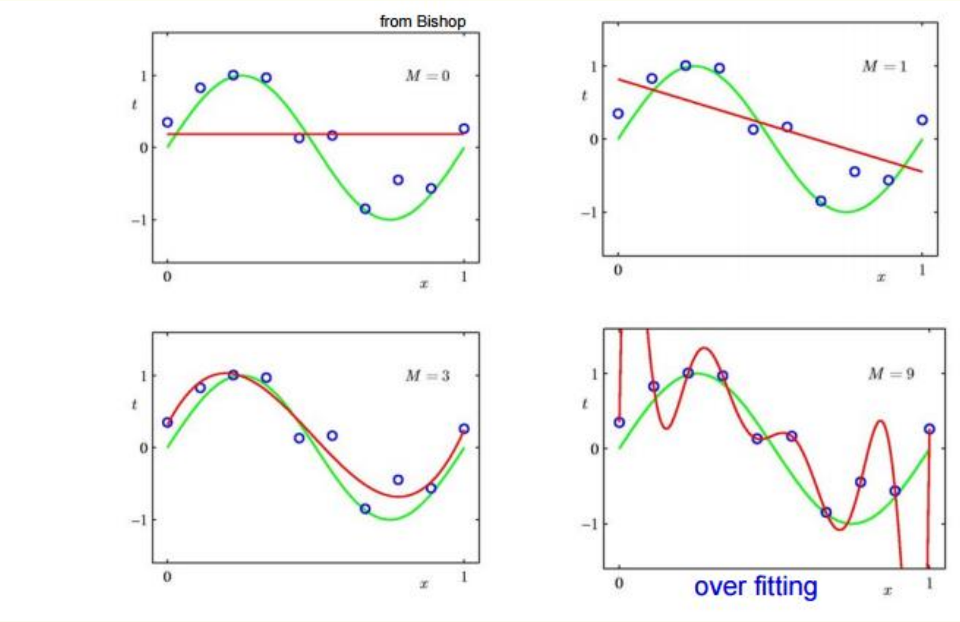
Generalized Linear Classifier

- Usually face severe overfitting
  - too many degrees of freedom
  - boundary can "curve" to fit to the noise in the data
- Regression example



Generalized Linear Classifier

- Helps to regularize by keeping **w** small
  - small **w** means the boundary is not as curvy
- Regression example



Polynomial Coefficients

	$M=0$	$M=1$	$M=3$	$M=9$
$w_0^\star$	0.19	0.82	0.31	0.35
$w_1^\star$		-1.27	7.99	232.37
$w_2^\star$			-25.43	-5321.83
$w_3^\star$			17.37	48568.31
$w_4^\star$				-231639.30
$w_5^\star$				640042.26
$w_6^\star$				-1061800.52
$w_7^\star$				1042400.18
$w_8^\star$				-557682.99
$w_9^\star$				125201.43

Generalized Linear Classifier

- Helps to *regularize* by keeping **w** small
  - small **w** means the boundary is not as curvy
- For example, add $\boldsymbol{\lambda}||\mathbf{w}||^2$ to the loss function
- Recall quadratic loss function

$$\mathbf{L} = \frac{1}{2}\Sigma_i || \mathbf{f}(\mathbf{x}^i, \mathbf{w}) - \mathbf{y}^i ||^2$$

- Regularized version

$$\mathbf{L} = \frac{1}{2}\Sigma_i || \mathbf{f}(\mathbf{x}^i, \mathbf{w}) - \mathbf{y}^i ||^2 + \boldsymbol{\lambda}||\mathbf{w}||^2$$

- Regression example, polynomial coefficients for degree M = 9
- With weight regularizer, gradient of loss function has a new term $-\alpha\lambda\mathbf{w}$

	small $\boldsymbol{\lambda}$	medium $\boldsymbol{\lambda}$	large $\boldsymbol{\lambda}$
$w_0^\star$	0.35	0.35	0.13
$w_1^\star$	232.37	4.74	-0.05
$w_2^\star$	-5321.83	-0.77	-0.06
$w_3^\star$	48568.31	-31.97	-0.05
$w_4^\star$	-231639.30	-3.89	-0.03
$w_5^\star$	640042.26	55.28	-0.02
$w_6^\star$	-1061800.52	41.32	-0.01
$w_7^\star$	1042400.18	-45.95	-0.00
$w_8^\star$	-557682.99	-91.53	0.00
$w_9^\star$	125201.43	72.68	0.01

Generalized Linear Classifier

- **λ** is a meta-parameter, cannot tune on training data
  - use validation or cross-validation to set it to a good value

- Consider polynomial of degree M=9 regression