IN THE NAME OF ALLAH

Neural Networks

Gradient Descent Learning Rule

Shahrood University of Technology
Hossein Khosravi

Homework #1

- Write a program for classification of samples of two classes using single layer perceptron:
- Samples are in 2D space
- Sample preparation
  - Randomly around a line
  - From 2 Gaussian distributions with different μ and σ
- Show the results in each iteration visually
- Use C++, MATLAB or other languages
- Optional: User can produce samples interactively
- Repeat the problem with 5 classes
- Due date: within 10 days

Gradient Descent Learning Rule

- Gradient Descent: Consider linear unit without threshold and continuous output *o* (not just –1,1)

$$o(X) = w_0 + w_1 x_1 + \dots + wm\, xm = W^{\mathrm{T}} X$$

- The squared error (where D is the set of training examples)
  - For an example (x,d) the error e(w) of the network is

$$e(W) = d - o(X) = d - \sum_{j=0}^{m} x_j w_j$$

  - And the squared error of (x,d) is

$$E(W) = \frac{1}{2} e^2$$

Gradient Descent Learning Rule

- The total squared error is

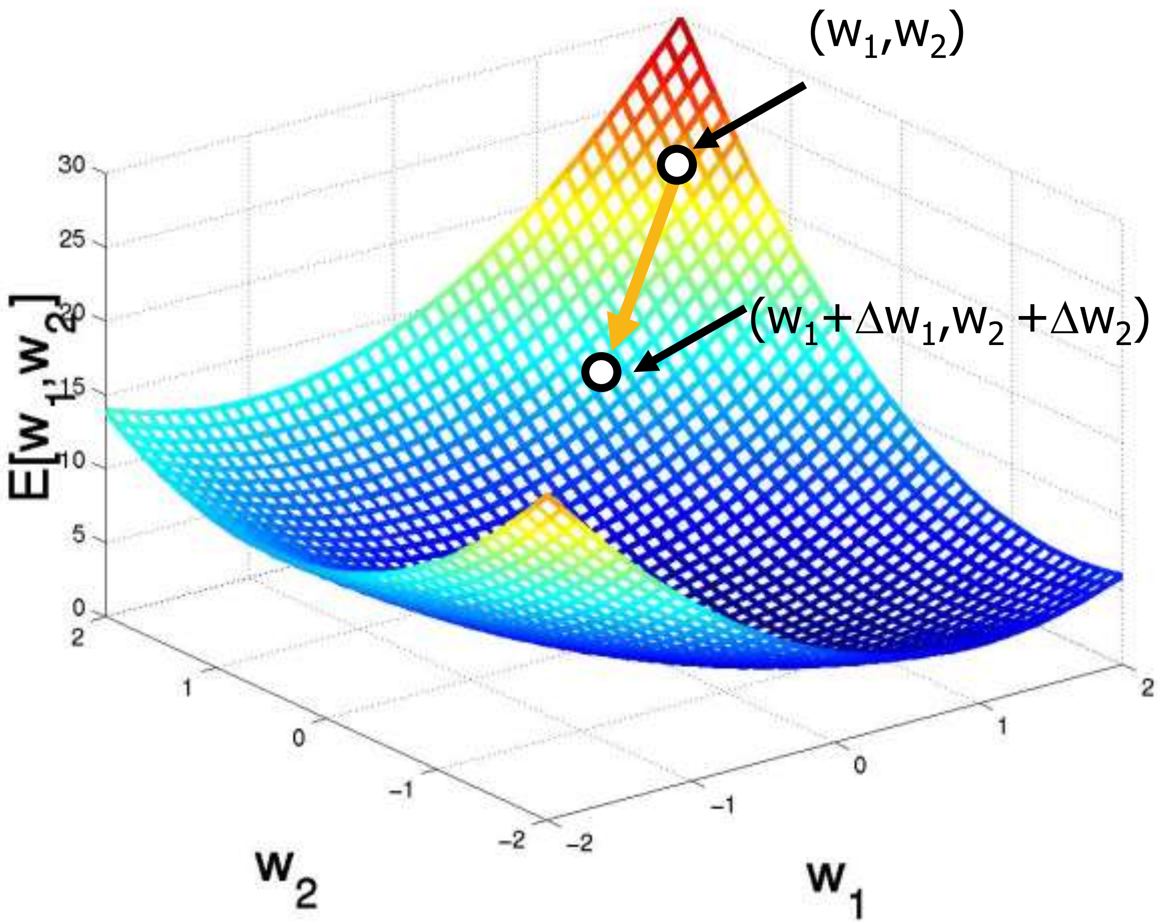
$$E(W) = E(w_1, \dots, wm) = \frac{1}{2} \sum_{(x,d) \in D} (d - o(x))^2$$

$$E(W) = E(w_1, \dots, wm) = \frac{1}{2} \sum_{(x,d) \in D} (d - WTX)^2$$

- Update the weight w_i such that $E(W) \rightarrow$ minimum

$$w_i(k+1) = w_i(k) + \Delta wi$$

Gradient Descent Learning Rule



Gradient Descent Learning Rule

- Start from an arbitrary point in the **weight space**

- The direction in which the error E of an example (as a function of the weights) is **decreasing most rapidly** is the opposite of the gradient of E:

$$-\nabla E(W) = -\left[\frac{\partial E}{\partial w_1}, \ldots, \frac{\partial E}{\partial w_m}\right]$$

- Take a small step (of size η) in that direction

$$\mathrm{w}(\mathrm{k}+1) = \mathrm{w}(\mathrm{k}) - \eta(\nabla\ \mathrm{E}(\mathrm{W}))$$

Gradient Descent Learning Rule

- Train the w_i's such that they minimize the squared error
  - $E(W) = E(w_1, \dots, wm) = \frac{1}{2} \sum_n (dn - on(x))^2$
- Gradient: $\nabla E(w) = [\frac{\partial E}{\partial w_0}, \dots, \frac{\partial E}{\partial w_m}]$

  $\Delta \boldsymbol{w} = -\eta \nabla \boldsymbol{E}(\boldsymbol{w})$

  $\Delta w_i = -\eta\, \partial E/\partial w_i = -\eta\, \partial/\partial w_i\; 1/2 \Sigma_n (d_n - o_n)^2$

  $= -\eta\, \partial/\partial w_i\; 1/2 \Sigma_n (d_n - \Sigma_i w_i x_{i,n})^2$

  $= -\eta\, \Sigma_n (d_n - o_n)(-x_{i,n})$

  $= \eta\, \Sigma_n (d_n - o_n) x_{i,n}$
- Gradient descent learning rule

$w_i(k+1) = w_i(k) + \Delta w_i = w_i(k) + \eta\, \Sigma_n (d_n - o_n) x_{in}$

Batch Learning Pseudo Code

Denoting a training example $(x(n), d_n)$ or $((x_{1n},\ldots x_{mn}),d_n)$ where $(x_{1n},\ldots,x_{mn})$ are the input values, and d_n is the desired output

- $k=1$, randomly initialize $w_i(k)$, calculate $E(W)$
- **While** ($E(W)$ unsatisfactory **&&** $k <$ max_iterations)
  - Initialize each Δw_i to zero
  - **For** each instance $(x(n), d_n)$ in *D* do
    - Calculate network output $o_n = \sum_i w_i(k)x_{in}$
    - **For** each weight dimension $w_i(k)$, $i=1,..,m$
      - $\Delta w_i = \Delta w_i + \eta(d_n - o_n)\, x_{in}$
    - **End For**
  - **End For**
  - **For** each weight dimension $w_i(k)$, $i=1,..,m$
    - $w_i(k+1)=w_i(k)+\Delta w_i$
  - **EndFor**
  - Calculate $E(W)$ based on updated $w_i(k+1)$
  - $k=k+1$
- **End While**

Example

- Consider the 2-dimensional training set $C_1 \cup C_2$,
- $C_1 = \{(1,1), (1, -1), (0, -1)\}$ with class label 1
- $C_2 = \{(-1,-1), (-1,1), (0,1)\}$ with class label 0
- Train a adaline on $C_1 \cup C_2$

Example – small η

- C1: {(1, 1, 1), (1, 1, -1), (1, 0, -1)}
- C2: {(1, -1,-1), (1, -1,1), (1, 0,1)}
- $\eta=0.1$
- $w_i(k+1)=w_i(k)+\Delta w_i$; $\Delta w_i= \eta\Sigma_n(d_n-o_n)x_{in}$
- Fill out this table sequentially (**First** pass):

Input	w(k)	d_n	o_n	$\eta(d_n-o_n)x_{in}$	Δw_i
(1, 1, 1)	(1,0, 0)	1	1	(0, 0, 0)	(0, 0, 0)
(1, 1, -1)	(1, 0, 0)	1	1	(0, 0, 0)	(0, 0, 0)
(1,0, -1)	(1, 0, 0)	1	1	(0, 0, 0)	(0, 0, 0)
(1,-1, -1)	(1, 0, 0)	0	1	(-0.1, 0.1, 0.1)	(-0.1, 0.1, 0.1)
(1,-1, 1)	(1, 0, 0)	0	1	(-0.1, 0.1, -0.1)	(-0.2, 0.2, 0)
(1, 0, 1)	(1, 0, 0)	0	1	(-0.1, 0, -0.1)	(-0.3, 0.2, -0.1)
	E(W)=3/2			w(k+1)	(0.7, 0.2, -0.1)

Example – small η

- C1: {(1, 1, 1), (1, 1, -1), (1, 0, -1)}
- C2: {(1, -1,-1), (1, -1,1), (1, 0,1)}
- η=0.1
- $w_i(k+1)=w_i(k)+\Delta w_i$; $\Delta w_i= \eta\Sigma_n(d_n-o_n)x_{in}$
- Fill out this table sequentially (**Second** pass):

Input	w(k)	d_n	O_n	$\eta(d_n-o_n)x_{in}$	Δw_i
(1, 1, 1)	(0.7, 0.2, -0.1)	1	0.8	(0.02, 0.02, 0.02)	(0.02,0.02,0.02)
(1, 1, -1)	(0.7, 0.2, -0.1)	1	1	(0,0,0)	(0.02,0.02,0.02)
(1,0, -1)	(0.7, 0.2, -0.1)	1	0.8	(0.02,0,-0.02)	(0.04,0.02,0)
(1,-1, -1)	(0.7, 0.2, -0.1)	0	0.6	(-0.06,0.06,0.06)	(-0.02,0.08,0.06)
(1,-1, 1)	(0.7, 0.2, -0.1)	0	0.4	(-0.04,0.04,-0.04)	(-0.06,0.12,0.02)
(1, 0, 1)	(0.7, 0.2, -0.1)	0	0.6	(-0.06,0,-0.06)	(-0.12,0.12,-0.04)
	E(W)=0.96/2			w(k+1)	(0.58,0.32,-0.14)

Note that unlike perceptron, output can take values other than 0 and 1

Example – small η

- C1: {(1, 1, 1), (1, 1, -1), (1, 0, -1)}
- C2: {(1, -1,-1), (1, -1,1), (1, 0,1)}
- η=0.1
- $w_i(k+1)=w_i(k)+\Delta w_i$; $\Delta w_i= \eta\Sigma_n(d_n-o_n)x_{in}$
- Fill out this table sequentially (**Third** pass):

Input	w(k)	d_n	O_n	$\eta(d_n-o_n)x_{in}$	Δw_i
(1, 1, 1)	(0.58,0.32,-0.14)	1	0.76	(0.024,0.024,0.024)	(0.024,0.024,0.024)
(1, 1, -1)	(0.58,0.32,-0.14)	1	1.04	(-0.004,-0.004,0.004)	(0.02,0.02,0.028)
(1,0, -1)	(0.58,0.32,-0.14)	1	0.72	(0.028,0,-0.028)	(0.048,0.04,0)
(1,-1, -1)	(0.58,0.32,-0.14)	0	0.4	(-0.06,0.06,0.06)	(-0.012,0.1,0.06)
(1,-1, 1)	(0.58,0.32,-0.14)	0	0.12	(-0.088,0.088,-0.088)	(-0.1,0.188,-0.028)
(1, 0, 1)	(0.58,0.32,-0.14)	0	0.44	(-0.056,0,-0.056)	(-0.156,0.188,-0.084)
	E(W)=0.5056/2			w(k+1)	(0.424,0.508,-0.224)

Gradient Descent Learning Rule

- Because the error surface contains only a single global minimum, this algorithm will converge to a weight vector with minimum error, regardless of whether the training examples are linearly separable, given a sufficiently small learning rate η

- If η is too large
  - Overstepping the minimum in the error surface

- Gradually reduce the value of η as the number of gradient descent steps grows

Example – large η

- C1: {(1, 1, 1), (1, 1, -1), (1, 0, -1)}
- C2: {(1, -1,-1), (1, -1,1), (1, 0,1)}
- η=1
- $w_i(k+1)=w_i(k)+\Delta w_i$; $\Delta w_i = \eta\Sigma_n(d_n-o_n)x_{in}$
- Fill out this table sequentially (**First** pass):

Input	w(k)	d	O	$\eta(d-o)x_i$	Δw_i
(1, 1, 1)	(1,0, 0)	1	1	(0, 0, 0)	(0, 0, 0)
(1, 1, -1)	(1, 0, 0)	1	1	(0, 0, 0)	(0, 0, 0)
(1,0, -1)	(1, 0, 0)	1	1	(0, 0, 0)	(0, 0, 0)
(1,-1, -1)	(1, 0, 0)	0	1	(-1, 1, 1)	(-1, 1, 1)
(1,-1, 1)	(1, 0, 0)	0	1	(-1, 1, -1)	(-2, 2, 0)
(1, 0, 1)	(1, 0, 0)	0	1	(-1, 0, -1)	(-3, 2, -1)
	E(W)=3/2			w(k+1)	(-2, 2, -1)

Example – large η

- C1: {(1, 1, 1), (1, 1, -1), (1, 0, -1)}
- C2: {(1, -1,-1), (1, -1,1), (1, 0,1)}
- $\eta=1$
- $w_i(k+1)=w_i(k)+\Delta w_i$; $\Delta w_i= \eta\Sigma_n(d_n-o_n)x_{in}$
- Fill out this table sequentially (**Second** pass):

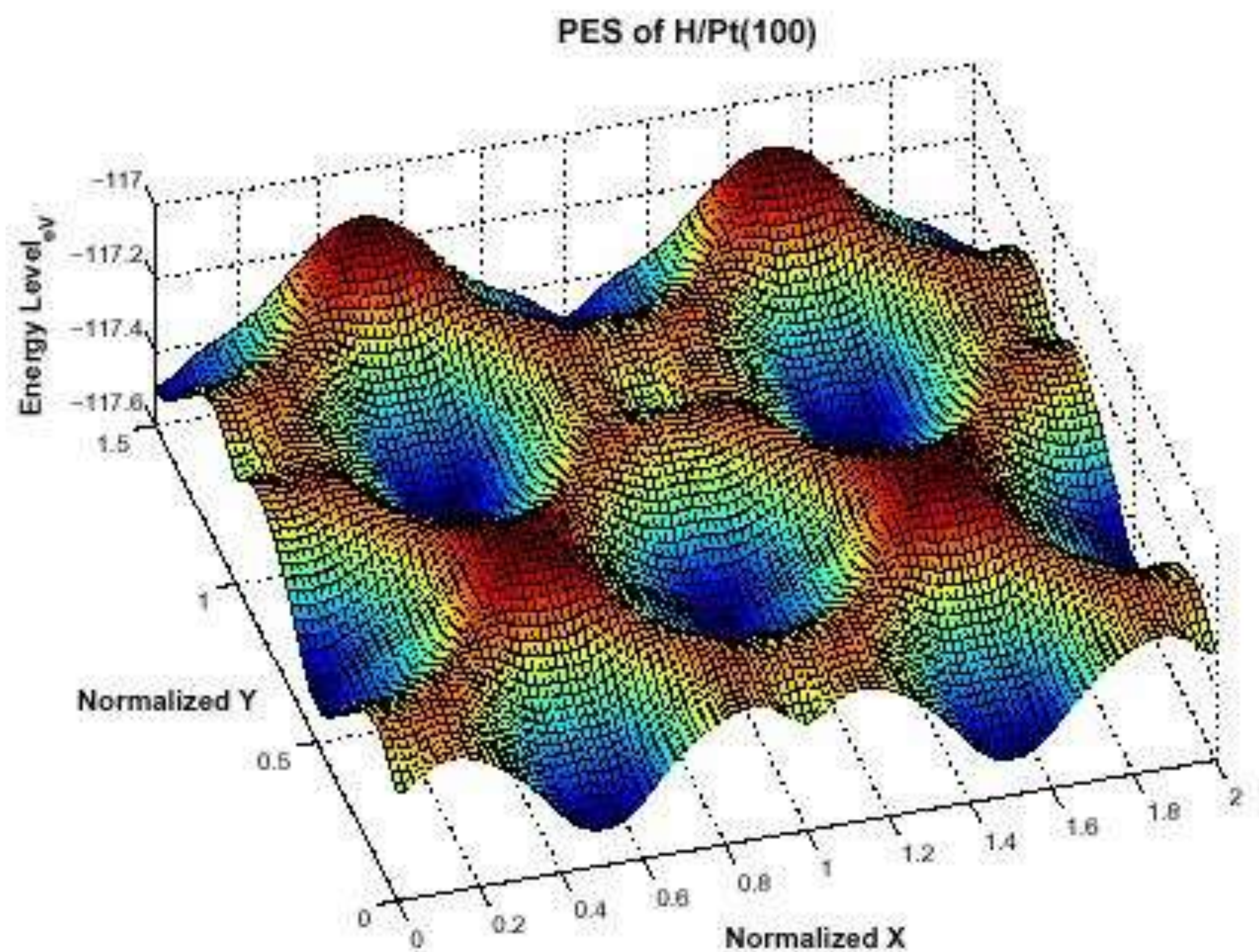
Input	w(k)	d	O	$\eta(d-o)x_i$	Δw_i
(1, 1, 1)	(-2,2, -1)	1	-1	(2, 2, 2)	(2, 2, 2)
(1, 1, -1)	(-2,2, -1)	1	1	(0, 0, 0)	(2, 2, 2)
(1,0, -1)	(-2,2, -1)	1	-1	(2, 0, -2)	(4, 2, 0)
(1,-1, -1)	(-2,2, -1)	0	-3	(3, -3, -3)	(7, -1, -3)
(1,-1, 1)	(-2,2, -1)	0	-5	(5, -5, 5)	(12, -6, 2)
(1, 0, 1)	(-2,2, -1)	0	-3	(3, 0, 3)	(15, -6, 5)
	E(W)=51/2			w(k+1)	(13, -4, 4)

Example – large η

- C1: {(1, 1, 1), (1, 1, -1), (1, 0, -1)}
- C2: {(1, -1,-1), (1, -1,1), (1, 0,1)}
- η=1
- $w_i(k+1)=w_i(k)+\Delta w_i$; $\Delta w_i= \eta\Sigma_n(d_n-o_n)x_{in}$
- Fill out this table sequentially (**Third** pass):

Input	w(k)	d	O	$\eta(d-o)x_i$	Δw_i
(1, 1, 1)	(13, -4, 4)	1	13	(-12,-12,-12)	
(1, 1, -1)	(13, -4, 4)	1	5	(-4,-4,4)	
(1,0, -1)	(13, -4, 4)	1	9	(-8,0,8)	
(1,-1, -1)	(13, -4, 4)	0	13	(-13,13,13)	
(1,-1, 1)	(13, -4, 4)	0	21	(-21,21,-21)	
(1, 0, 1)	(13, -4, 4)	0	17	(-17,0,-17)	(-75,18,-25)
	E(W)=1123/2			w(k+1)	(-62,14,-21)

Local minimum



Incremental Stochastic Gradient Descent

- Batch mode : Gradient descent
  - $w(k+1)=w(k) - \eta\nabla E_D(W)$ **over the entire data D**
  - $E_D(W)=1/2\Sigma_n(d_n-o_n)^2$
- Incremental mode: Gradient descent
  - $w(k+1)=w(k) - \eta\nabla E_n(W)$ **over individual training examples**
  - $E_n(W)=1/2\ (d_n-o_n)^2$
- Incremental Gradient Descent can approximate Batch Gradient Descent arbitrarily close if η is small enough

Weights Update Rule: incremental mode

- Computation of Gradient(E):

$$\frac{\partial E(\mathrm{W})}{\partial \mathrm{w}} = \frac{\partial}{\partial \mathrm{w}}\left(\frac{1}{2}(d_n - o_n)^2\right) = \frac{\partial}{\partial \mathrm{w}}\left(\frac{1}{2}(d_n - w^T x(n))^2\right)$$
$$= -(d_n - o_n)x(n)$$

- Delta rule (**AdaLine**: Adaptive Linear Elements) for weight update:

$$\mathrm{w}(\mathrm{k}+1) = \mathrm{w}(\mathrm{k}) - \eta \frac{\partial E(W)}{\partial w}$$
$$\mathrm{w}(\mathrm{k}+1) = \mathrm{w}(\mathrm{k}) + \eta(\mathrm{d_n} - o_n)x(n)$$

Delta Rule (AdaLine) learning algorithm

k=1;

initialize $w_i(k)$ randomly; Calculate $E_D(W)$

while ($E_D(W)$ unsatisfactory **AND** k<max_iterations)

Select an example $(x(n), d_n)$

$$\Delta w_i = \eta(d_n - o_n)x_{in}$$

$$w_i(k+1) = w_i(k) + \Delta w_i$$

Calculate $E_D(W)$

k = k+1;

end-while;

Example – incremental mode

C1: {(1, 1, 1), (1, 1, -1), (1, 0, -1)}
C2: {(1, -1,-1), (1, -1,1), (1, 0,1)}
η=0.1
Fill out this table sequentially (First pass):
$w_i(k+1)=w_i(k)+\eta(d-o)x_i$

Input	$w_i(k)$	d	O	$\eta(d-o)x_i$	$w_i(k+1)$
(1, 1, 1)	(1,0, 0)	1	1	(0, 0, 0)	(1, 0, 0)
(1, 1, -1)	(1, 0, 0)	1	1	(0, 0, 0)	(1, 0, 0)
(1,0, -1)	(1, 0, 0)	1	1	(0, 0, 0)	(1, 0, 0)
(1,-1, -1)	(1, 0, 0)	0	1	(-0.1, 0.1, 0.1)	(0.9, 0.1, 0.1)
(1,-1, 1)	(0.9, 0.1, 0.1)	0	0.9	(-0.09, 0.09, -0.09)	(0.81,0.19,0.01)
(1, 0, 1)	(0.81,0.19,0.01)	0	0.82	(-0.082, 0, -0.082)	(0.728, 0.19, -0.072)

Perceptron Learning Rule vs. Gradient Descent Rule

Perceptron learning rule guaranteed to succeed if

- Training examples are linearly separable
- Sufficiently small learning rate η

Gradient descent learning rules

- Guaranteed to converge to hypothesis with minimum squared error
- Given sufficiently small learning rate η
- Even when training data contains noise
- Even when training data not separable by H

Comparison of Perceptron and Adaline

	Perceptron	Adaline
Architecture	Single-layer	Single-layer
Neuron model	Non-linear	linear
Learning algorithm	Minimize number of misclassified examples	Minimize total squared error
Application	Linear classification	Linear classification, regression