Adaline

COMP4211

THE DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
計算機科學及工程學系

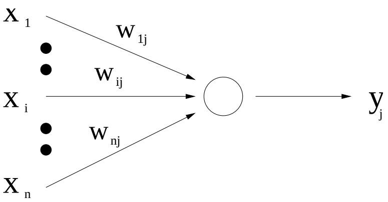
Outline

1. Model
2. Gradient Descent
3. SGD

Introduction

Adaline (Adaptive Linear Element)

- a feed-forward network with one layer of adjustable weights connected to one or more linear units (as output units)



$$o = \sum_{i=0}^{n} w_i x_i$$

- in statistics, this is called linear regression

Learning the Adaline

- target output for training pattern d: t_d
- output (of the linear unit) for training pattern d: $o_d(\vec{w}) = o_d$

square error on the training set:

$$E(\vec{w}) = \sum_{d \in D} E_d = \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2$$

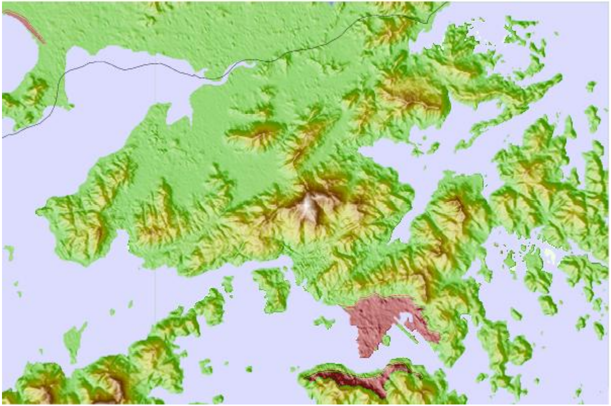
how to find $\vec{w}$ that minimizes $E(\vec{w})$?

in linear regression

- $\vec{w}$ can be obtained in closed-form
- here we present another approach, just to warm up for MLP learning
- MLP is regarded as a tool of nonlinear regression

Learning by Gradient Descent

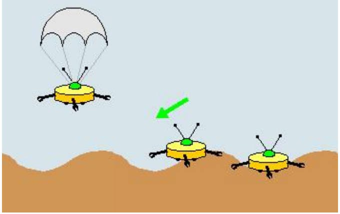
Motivating Example: How to go to Tai Mo Shan?



- start at any point and keep going uphill
- use gradient descent to search the space of possible weight vectors to find the weights that minimizes E

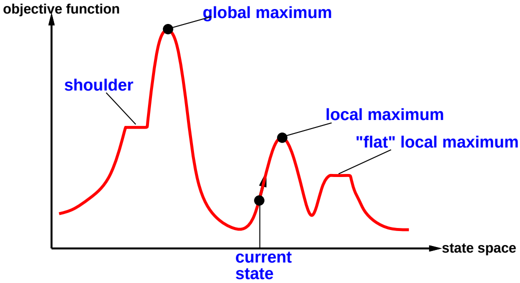
Finding the Weight

- start at any point and keep going downhill



Error Surface $E(\vec{w})$

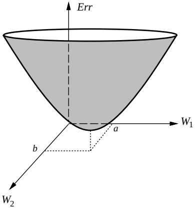
- in general, the error surface can be very complicated
- useful to consider this as a landscape



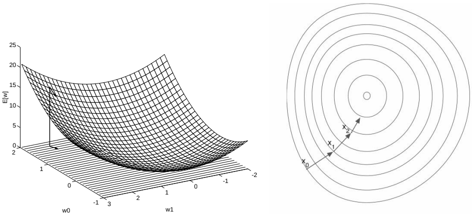
- can get stuck in locally optimal solutions

Error Surface $E(\vec{w})$...

- but here, $E(\vec{w}) = \frac{1}{2}\sum_{d \in D}(t_d - o_d)^2$ with $o = \sum_{i=0}^{n} w_i x_i$ is of the form



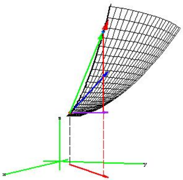
- a global minimum!

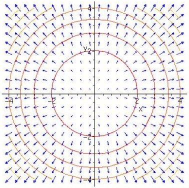


Gradient Descent

$$\vec{w} \leftarrow \vec{w} + \Delta\vec{w}$$

gradient $\nabla E[\vec{w}]$ at $\vec{w}$:
$\left[\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \cdots \frac{\partial E}{\partial w_n}\right]$





move $\vec{w}$:

- direction: opposite to $\nabla E[\vec{w}]$
- magnitude: a small fraction of $\nabla E[\vec{w}]$

$$\Delta\vec{w} = -\eta\nabla E[\vec{w}]$$

$$\Delta w_i = -\eta\frac{\partial E}{\partial w_i}$$

Math

$$
\begin{aligned}
\frac{\partial E}{\partial w_i} &= \frac{\partial}{\partial w_i} \frac{1}{2} \sum_d (t_d - o_d)^2 \\
&= \frac{1}{2} \sum_d \frac{\partial}{\partial w_i} (t_d - o_d)^2 \\
&= \sum_d (t_d - o_d) \frac{\partial}{\partial w_i} (t_d - \vec{w} \cdot \vec{x_d}) \\
&= \sum_d (t_d - o_d)(-x_{i,d}) \\
\Delta w_i &= -\eta \frac{\partial E}{\partial w_i} = \eta \sum_d (t_d - o_d) x_{i,d}
\end{aligned}
$$

Delta Rule (LMS rule, Adaline rule, Widrow-Hoff rule)

begin
  initialize each w_i to some small random value;
  repeat
    initialize each Δw_i to zero;
    for *each* $\langle \vec{x}, t \rangle$ *in the training set* D **do**
      input instance $\vec{x}$ to the unit and compute output o;
      for *each linear unit weight* w_i **do**
        $\Delta w_i \leftarrow \Delta w_i + \eta(t - o)x_i$;
      end
    end
    for *each linear unit weight* w_i **do**
      $w_i \leftarrow w_i + \Delta w_i$;
    end
  until *termination condition is met*;
end

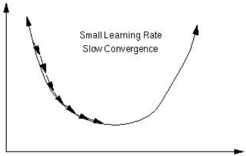
Termination Conditions

- when $||\Delta\vec{w}||$ is smaller than a threshold value
- when the number of iterations has reached a preset maximum

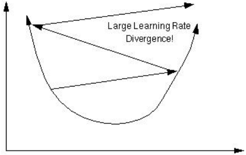
Convergence

- error surface contains only a single global minimum
- the delta rule will converge to a weight vector with minimum error if an appropriate learning rate is chosen

sufficiently small learning rate η

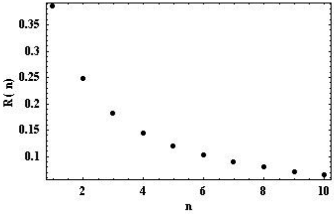


η is too large $\rightarrow$ may over-step the minimum in the error surface



Learning Rate Schedule

gradually reduce η as the number of gradient descent steps grows



A practical trick

- perform experiments using a small subset of the training set
- when the algorithm performs well on this small subset, keep the same η, and let it run on the full training set

Stochastic Gradient Descent

Batch Learning

$$\Delta w_i = \eta \sum_d (t_d - o_d) x_{i,d}$$

- sum the gradient over the whole data set

on big data sets, this can be very expensive

- after reading all the records, you can move one step (iteration)
- then repeat for every step

what can you do?

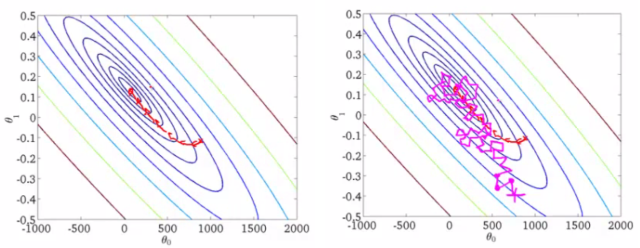
Stochastic Gradient Descent (SGD)

- update the weights after each individual sample
  - requires fewer computation per weight update step

begin
  Initialize each w_i to some small random value;
  repeat
    for *each* $\langle \vec{x}, t \rangle$ *in the training set* D **do**
      input instance $\vec{x}$ to the unit and compute output o ;
      for *each linear unit weight* w_i **do**
        $w_i \leftarrow w_i + \Delta w_i = w_i + \eta(t - o)x_i$;
      end
    end
  until *termination condition is met*;
end

Example

left: batch: right: stochastic



- stochastic gradient descent every iteration is much faster
- you “generally” move in the right direction, but not always
- a smaller step size has to be used

if you have a truly massive dataset

- it is possible that a single pass over the data can produce a perfectly good network
- in contrast, for batch gradient descent, one always has to make multiple passes over the data

Tradeoff on the Number of Samples in Each Iteration

- use all samples in each iteration: accurate but slow
- use 1 sample in each iteration: fast but highly variable
- use b samples in each iteration (mini-batch)
  - b: mini-batch size (e.g., $b = 128$)
  - just like batch, except we use tiny batches
  - do not have to update parameters after every sample, and do not have to wait until you cycled through all the data