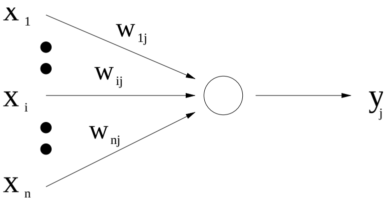
Adaline

COMP4211

Adaline (Adaptive Linear Element)

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



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

symbols

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

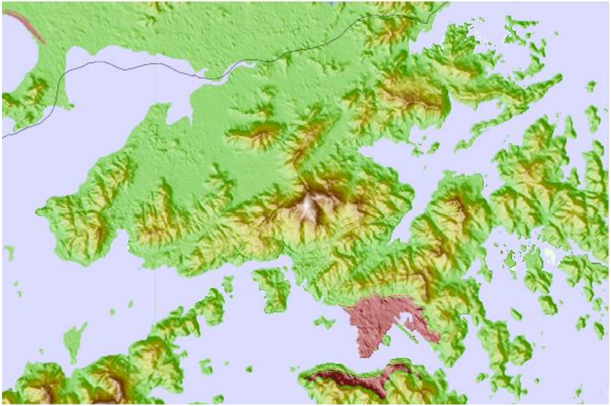
squared training error: $E(\vec{w}) = \sum_{d \in D} E_d = \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2$

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

Adaline and Linear Regression

- in statistics, adaline is called 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

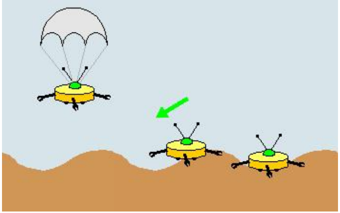
How to go to Tai Mo Shan?



- start at any point and keep going uphill

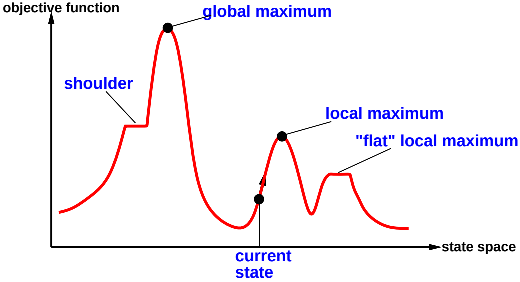
Finding the Weight

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



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

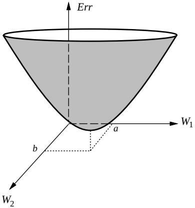
- in general, the error surface can be very complicated
- useful to consider this as a landscape



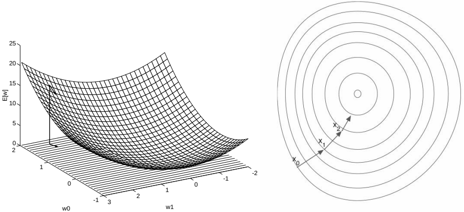
- can get stuck in locally optimal solutions

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

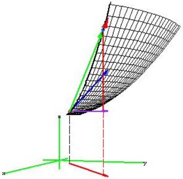


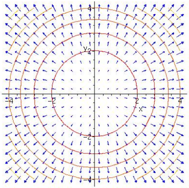
- a global minimum!



Gradient Descent

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 $\nabla E[\vec{w}]$

$$
\begin{aligned}
\vec{w} &\leftarrow \vec{w} + \Delta\vec{w} \\
\Delta\vec{w} &= -\eta \nabla E[\vec{w}] \\
\Delta w_i &= -\eta \frac{\partial E}{\partial w_i}
\end{aligned}
$$

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 ⟨x⃗, t⟩ in the training set D do
            input instance x⃗ to the unit and compute output o;
            for each linear unit weight w_i do
                Δw_i ← Δw_i + η(t − o)x_i ;
            end
        end
        for each linear unit weight w_i do
            w_i ← w_i + Δw_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

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

- you cannot store the whole data set in memory $\Rightarrow$ need to read into memory from disk
- after reading all the records, you can move one step (iteration)
- then repeat for every step
- take a long time to converge especially because disk I/O is typically a system bottleneck

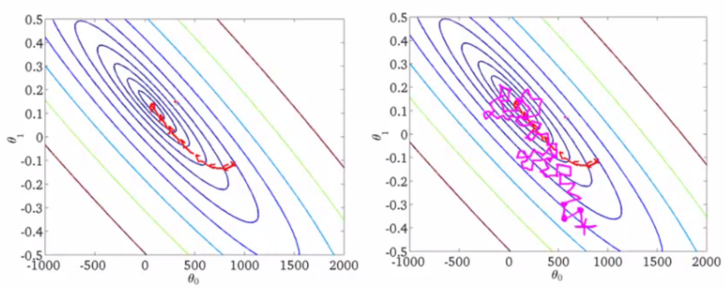
what can you do?

Stochastic Gradient

- update the weights after each individual example
  - 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



- 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

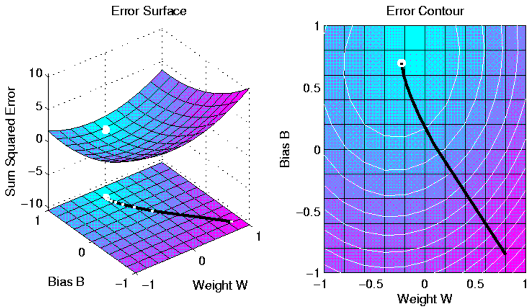
Mini-Batch

- batch gradient descent: use all examples in each iteration
- stochastic gradient descent: use 1 example in each iteration
- mini-batch gradient descent: use b examples in each iteration
  - b: mini-batch size (e.g., $b = 128$)
  - just like batch, except we use tiny batches
  - do not have to update parameters after every example, and do not have to wait until you cycled through all the data
  - often work faster than stochastic gradient descent

Convergence

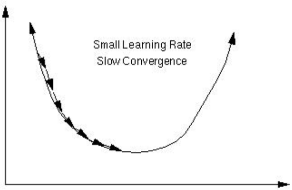
Because the error surface contains only a single global minimum, the delta rule will converge to a weight vector with minimum error if an appropriate learning rate is chosen

- even when training data contains noise / not separable by H

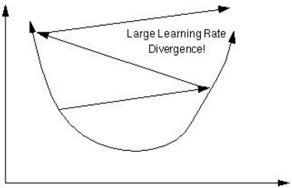


Overshooting

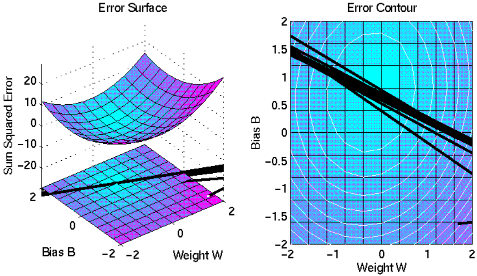
Condition: sufficiently small learning rate η



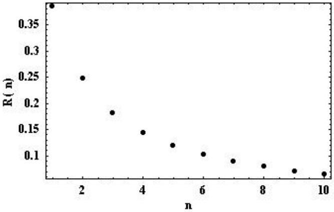
η is too large $\rightarrow$ may over-step the minimum in the error surface



Overshooting...



Solution: gradually reduce η as the number of gradient descent steps grows



Some Practical Tricks

Setting η using the training set

- 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

Some Practical Tricks...

Check the gradients using finite difference

- pick an example (x_d, t_d)
- compute the objective $E(\vec{w}; (x_d, t_d))$ for the current w
- compute the gradient $g = \frac{\partial E(\vec{w}; (x_d, t_d))}{\partial w_i}$
- apply a slight perturbation to w_i:
  $\vec{w}' = \vec{w} + [0, \ldots, 0, \delta, 0, \ldots, 0]$
- compute the new objective $E(\vec{w}'; (x_d, t_d))$ and verify that
  $E(\vec{w}'; (x_d, t_d)) \simeq E(\vec{w}; (x_d, t_d)) + \delta g$
- repeat the procedure for many examples (x_d, t_d), many perturbations δ and many initial weights $\vec{w}$