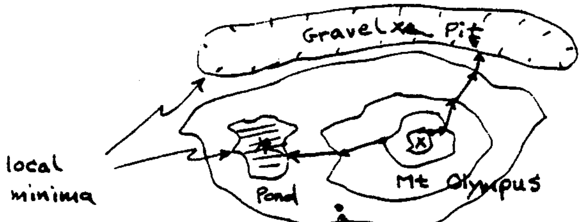
1 May 1988 Gradient Descent – Introduction

Neil E Cotter

ex: steepest path = fastest way down

- *Water flows in steepest path down a mountain*

ex: Given topo map find a local depression (i.e. *local minima*)



3000 ft deep tunnel (global minimum)

Start at pt on mountain

Follow gradient = steepest path

Note: 1) Sol'n depends on starting point. — *• Could end up in pond or in gravel pit*

2) Sol'n " " step size in direction of gradient ... — *• Could step right over the 3000 ft deep mine shaft*

3) Sol'n may not be global min

Gradient descent minimizes result (e.g. altitude) (or $o(w_{ij})$)
with respect to variable parameters (e.g. (attitude, longitude). (or weights w_{ij})

Mathematics:

def: gradient of $f(x_1, \ldots, x_n) \equiv \nabla f \equiv \begin{bmatrix} \partial f/\partial x_1 \\ \vdots \\ \partial f/\partial x_n \end{bmatrix}$

- *Multidimensional derivative*

ex: multi-dim Taylor series — *• See Duda & Heart p.140*

switch order

$$f(x_1+\epsilon_1, \ldots, x_n+\epsilon_n) = f(x_1, \ldots, x_n) + \nabla f \circ (\epsilon_1, \ldots, \epsilon_n) + \ldots$$

- *∇f is a function of $x_1, \ldots, x_n$.*
- *∇f is a vector that has a different value at each point $(x_1, \ldots, x_n)$*

Plot $f(x_1, \ldots, x_n)$ as surface in n-dim space. ∇f always points in direction of steepest slope. *• True at every pt $(x_1, \ldots, x_n)$*

May 1988 Gradient Descent - Introduction (cont.)

Neil E Cotter

$$\vec{\omega}_{new} = \vec{\omega}_{old} - \rho \nabla f(\vec{\omega})$$

ex: Perceptron Proportional Increment Training Rule

$$\text{Let } f(\vec{v}) = -\vec{\eta}_k \cdot \vec{v} \qquad \begin{aligned} \vec{v} &= (\omega_0, \omega_1, \ldots, \omega_n) \\ \vec{\eta} &= (1, x_1, \ldots, x_n) \end{aligned}$$

$$\text{Then } \nabla f(\vec{v}) = \nabla\underbrace{(-\omega_0 \cdot 1 - \omega_1 \cdot x_1 - \ldots - \omega_n x_n)}$$

$$= \begin{pmatrix} \frac{\partial}{\partial \omega_0}(\quad " \quad) \\ \frac{\partial}{\partial \omega_1}(\quad " \quad) \\ \vdots \\ \frac{\partial}{\partial \omega_n}(\quad\quad) \end{pmatrix}$$

$$= \begin{pmatrix} -1 \\ -x_1 \\ \vdots \\ -x_n \end{pmatrix} = -\vec{\eta}$$

$$\vec{v}_{new} = \vec{v}_{old} - \rho(-\vec{\eta}) = \vec{v}_{old} + \rho\vec{\eta}$$

or

$$\omega_{0\,new} = \omega_{0\,old} + \rho$$

$$\vec{\omega}_{new} = \vec{\omega}_{old} + \rho\,\vec{x} \qquad \begin{aligned} \vec{\omega} &= (\omega_1, \ldots, \omega_n) \\ \vec{x} &= (x_1, \ldots, x_n) \end{aligned}$$

- In class we used $\rho = 1$

ex: Delta Rules

$$\vec{w}_{new} = \vec{w}_{old} + \rho\,\underbrace{(V_{out\,(desired)} - V_{out})}_{\equiv\ \delta}\,\vec{x}$$

- Is proportional increment training for perceptrons.
- Works on various types of neural networks.