Recall the situation, the Deep, feed forward neural net (also called MLP = multi layered perceptron)

has the form

$$F(x,\eta) = F_L \circ \cdots \circ F_1(x) \quad \text{with each}$$

$$F_i(x) = \sigma(A_i x + \vec{b}_i)$$

- The parameters are $\eta = A_1, \ldots, A_L, b_1, \ldots, b_L$
- The training data is $x_1, \ldots, x_N$ with correct output $y_1, y_2, \ldots, y_N$
- The loss or error or objective function is (simplest version)

$$\Phi(\eta) = \frac{1}{N} \sum_{i=1}^{N} \| F(x_i, \eta) - y_i \|^2$$

- Learning consists of adjusting the parameters η so that the error diminishes.

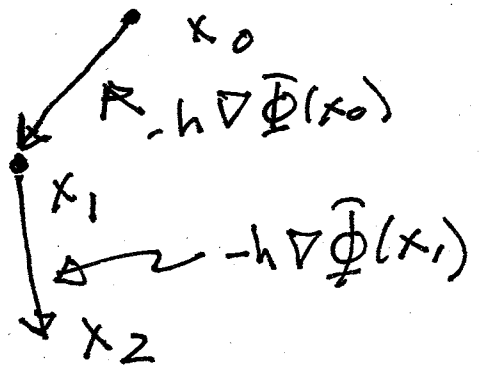
- To diminish Φ the first step is to find the direction we should move from $\vec{x}$ so that Φ gets smaller
- We had to next Theorem from Calculus.

Summary: Given $\Phi: \mathbb{R}^n \to \mathbb{R}$ at a point $\vec{x} \in \mathbb{R}^n$ the direction of maximal increase of Φ is given by $\nabla \Phi(x)$ and the direction of maximal decrease is given by $-\nabla \Phi(x)$. Further, for a level set $L_c = \{ \vec{x} \in \mathbb{R}^n : \Phi(\vec{x}) = c \}$ at a point $\vec{x}$ in L_c, $\nabla \Phi(\vec{x})$ is perpendicular to the level set [provided level set is nice at x, no corners, etc]

The idea of gradient descent is to take a step of size h in the direction in which Φ is decreasing the most, i.e. in the direction $-\nabla\Phi(x)$

OR $x_0 =$ initial position

$$x_1 = x_0 + h\left(-\nabla\Phi(x_0)\right)$$



we repeat

$$x_2 = x_1 + h\left(-\nabla\Phi(x_1)\right)$$

$$\vdots$$

or as a loop

- Input: x_0 and stepsize h and stop criteria
- Requires: subroutine to compute $-\nabla\Phi(x)$

- for $i = 1$ to n

$$x_i = x_{i-1} - h\,\nabla\Phi(x_{i-1})$$

end

- Two questions
  - how do we choose h?
  - when do we halt?

- Halting is a bit more straight forward at first

- Standard things are

  - $|x_{i+1} - x_i| < \varepsilon$, so the iteration isn't making much progress. Note that since

    $$x_i = x_{i-1} - h\nabla\Phi(x_{i-1})$$

    This is equivalent to ~~choosing~~ halting when

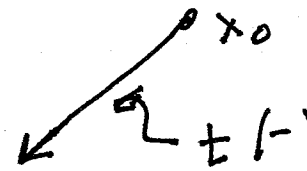
    $\|\nabla\Phi(x_i)\|$ is small

  - $|\Phi(x_{i+1}) - \Phi(x_i)| < \varepsilon$, so the function values aren't changing much

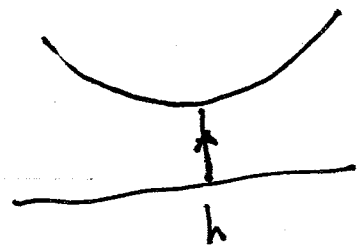
- But we could be in a gradually sloping valley or at a local min ....

- Picking h (also called the learning rate) has many aspects.

- The first idea is to do a line search



Let $g(t) = \Phi(x_0 - t\nabla\Phi(x_0))$ and then use a 1-D routine to minimize g as a function of t



and let $h = \text{argmin}(g(t))$

- Even with line seach gradient descent can get caught in "zig-zags"

(see wikipedia figures)

- ~~There are~~ Another idea is to choose different h's for different directions

$$x_1' = x_1 - h_1 \frac{\partial \Phi}{\partial x_1}(x) \qquad \text{where } x_L \to x_L' \text{ is}$$

$$x_2' = x_2 - h_2 \frac{\partial \Phi}{\partial x_1}(x) \qquad \text{the update.}$$

- yet another to choose directions other than $\nabla \Phi$ (conjugate gradients)

- There are many other ways to find $\min \Phi$ in the vast literature of Optimization Theory
- Some methods using higher derivatives (the Hessian $H\Phi$) or more evaluation of Φ and $\nabla\Phi$
- But since ~~Each~~ in a Deep Learning net each evaluation of Φ and $\nabla\Phi$ is very computationally expensive so some variant of Gradient Descent is used
- In the next Lecture we will discuss a special method to compute $\nabla\Phi$

- One last variant of Gradient Descent is important to mention

- Recall we have training set $x_1, \ldots, x_N$ with correct output $y_1, \ldots, y_N$ where $N = 10{,}000$ is possible

- The network's input-output function is

$$F(x, m) = F_L \circ \ldots \circ F_1(x)$$

where each $F_i = \sigma(A_i \vec{x} + \vec{b}_i)$ and the parameters are $m = (A_1, \ldots, A_L, b_1, \ldots, b_L)$

- and we have a loss or error function

$$\Phi(m) = \frac{1}{N} \sum_{i=1}^{N} \| F(x_i, m) - y_i \|^2$$

- Notice that one evaluation of Φ requires N evaluations of F not to say how expensive computing $\nabla \Phi$
- Stochastic Gradient Descent takes advantage of the fact that Φ is a sum of errors, one for each x_i from the training set
- This proceeds by picking a minibatch from the training data $\bar{x}_1, \ldots, \bar{x}_K$ and then computy $\bar{\Phi}(\eta) = \frac{1}{K} \sum_{i=1}^{K} \| F(\bar{x}_i, \eta) - \bar{y}_i \|^2$ and using this to compute $\nabla \bar{\Phi}$ and using Gradient Descent to update η.
- Then repeat with a new, radomnly chosen minibatch of data.