CIS501 – Lecture 17

Woon Wei Lee

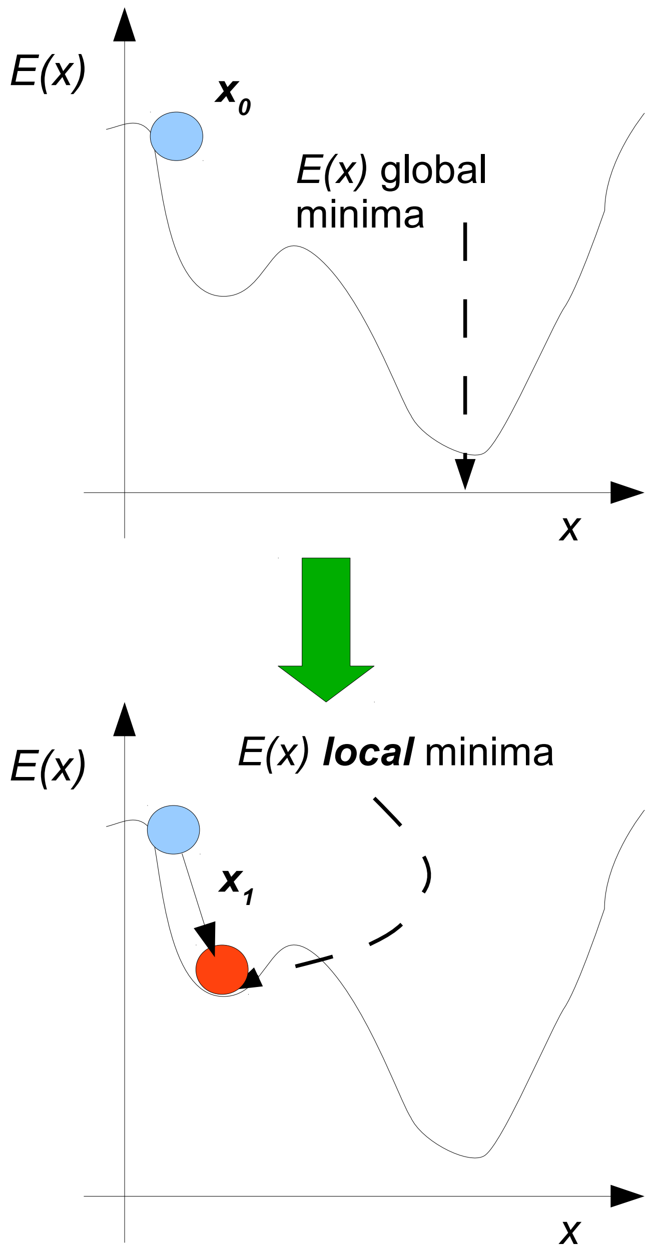
Fall 2013, 10:00am-11:15am,
Sundays and Wednesdays

Masdar INSTITUTE

For today:

- Problems with gradient descent
- Proposed solutions
  - Momentum
  - Levenberg-Marquadt
- Presentations
  - Nengbao Lin
  - Yanan Xiao

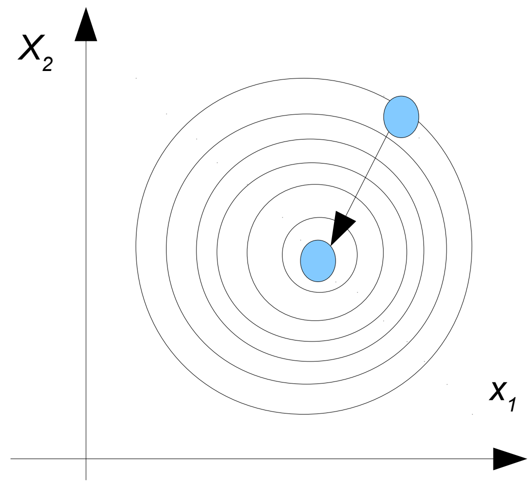
MLP challenges - Local minima

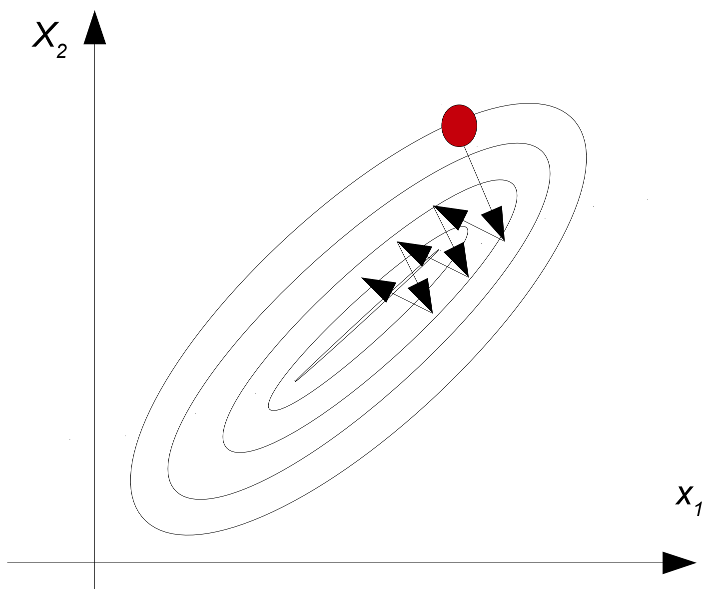


- **One of the most common problems in gradient based learning:**
  - In the previous example, the error function is “smooth” → easy to learn
  - In most cases, error function can be “bumpy”
  - Depending on the size of the learning step..
    - If step size is big enough, local minima may be avoided, but precision is reduced – convergence may be very slow..
    - If step size is too small, difficult to avoid local minima..
- **For complex multivariate error surfaces, local minima exceedingly common**

MLP challenges – slow convergence

- **For a smooth error surface, gradient descent works great.. in one dimension!**
  - For higher dimensional spaces, the curse of dimensionality factors in again..
  - “Shape” of the error surface is an important factor...
- **Arrows in graphs on right indicate direction of gradient term**
  - First graph → circular, bowl shaped error function..
  - Error gradient points directly to the global minimum – convergence is fast
  - Second graph → “elongated” error surface
  - Direction of gradient terms are almost perpendicular to the direction that we need to be heading!
- **Result: convergence using standard backpropagation - extremely slow! :-(**





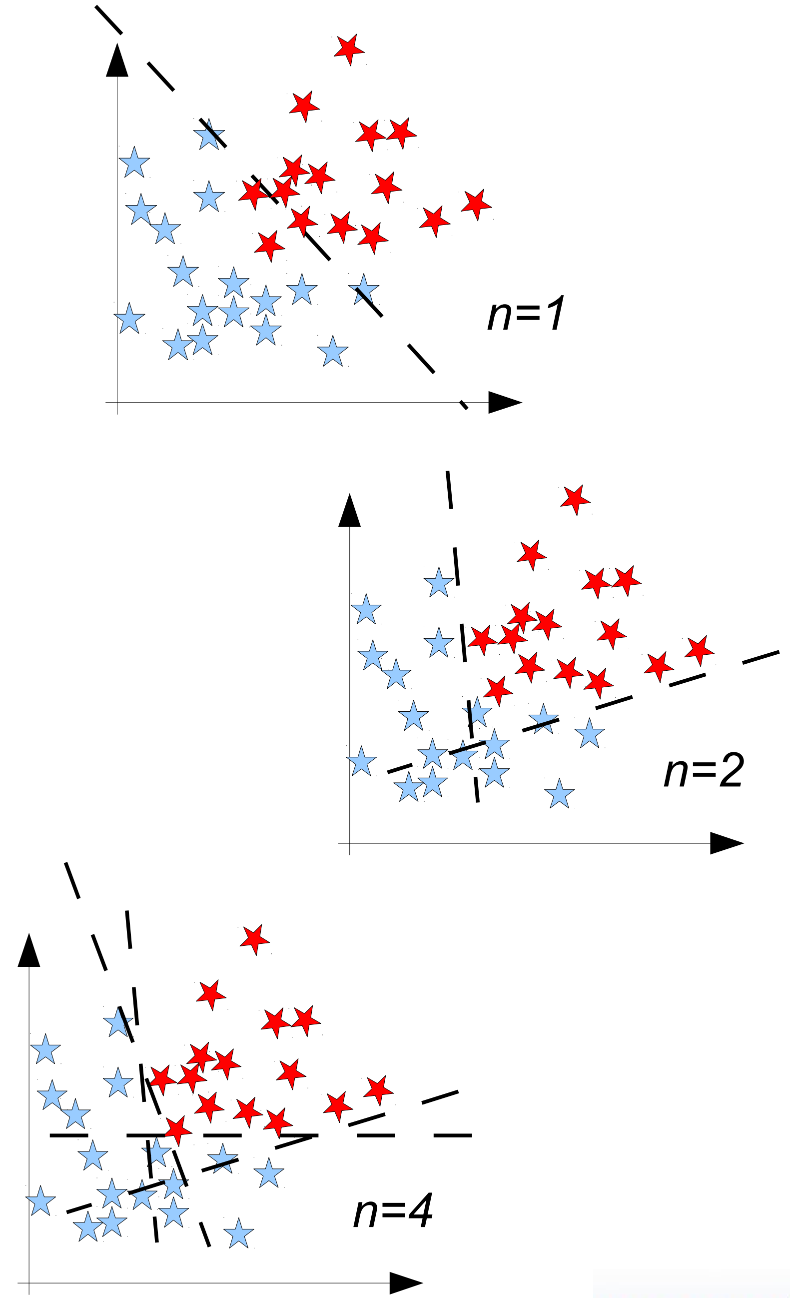
Masdar INSTITUTE

(challenges cont'd) – Over-fitting

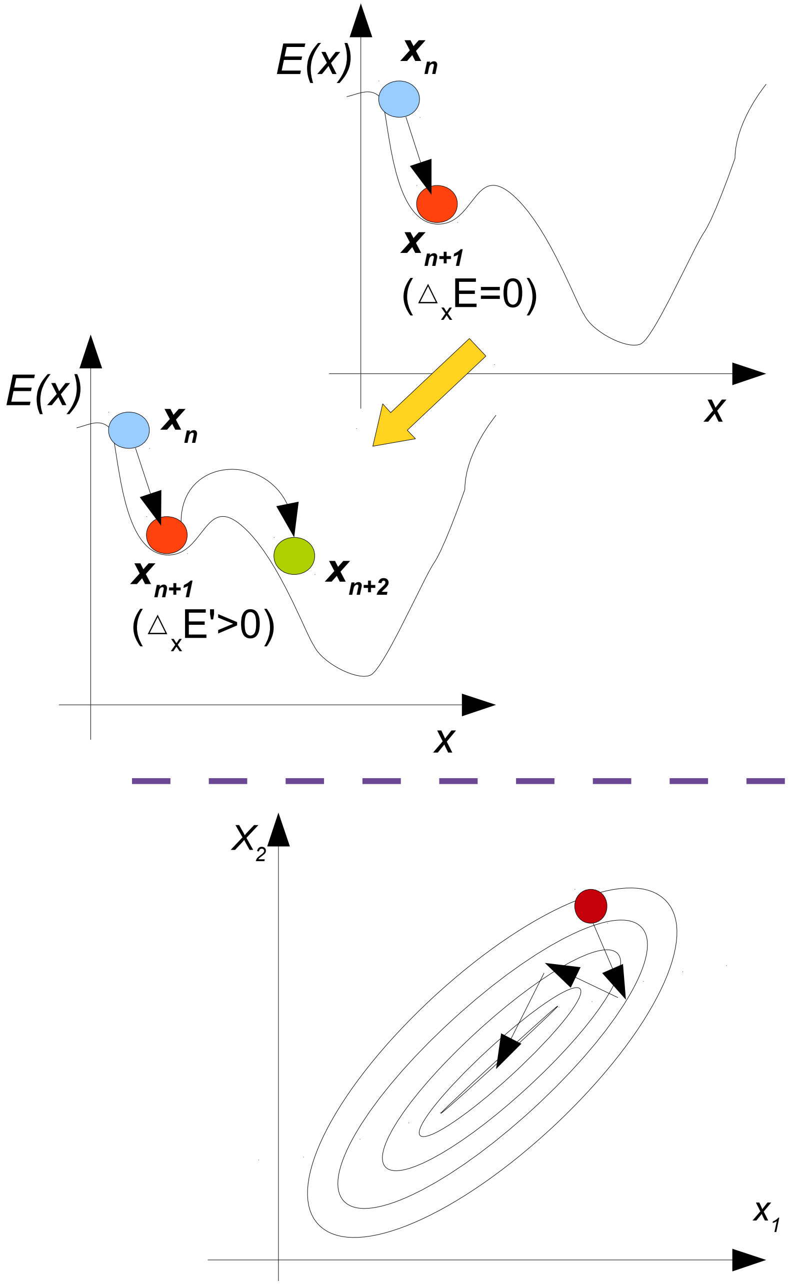
- **For an MLP, complexity is determined by *number of hidden units***
  - For classification: the number of hidden units determines the number of possible decision boundaries

    → the more boundaries, the more “nonlinear” the final boundary
  - For regression: the number of hidden units determines the number of basis functions

    → i.e. more “bends” in the regression curve
  - As with decision trees, trade-off between expressiveness, and overfitting of the data
- **No simple way to determining the number of hidden units**
  - Evaluate accuracy/ROC curves for a variety of network structures
  - Choose best one..!
  - Regularization



Gradient descent with momentum



- **Simple way which addresses both the local minima problem, and the problem of slow convergence**
  - Modification of the gradient term as follows:

$$w(n+1)=w(n)-\Delta_w{}'E$$

$$\Delta_w{}'E=\frac{\eta\cdot dE(w)}{dw}-\mu\cdot(w(n)-w(n-1))$$

  - In effect, a portion of the previous update term is added to the current term at each step
- **This has two effects:**
  - Momentum term helps weights update to “overshoot” local minima locations
  - Incorporation of a momentum term helps to enforce update directions that are consistent

    (helps with slow convergence problem)

Levenberg-Marquadt Algorithm

- **The momentum based gradient descent can be effective but has shortcomings:**
  - Requires the setting of arbitrary terms η and μ
  - In practice, it is an improvement over standard gradient descent but not a huge improvement
- **More efficient algorithms have been developed**
  - Utilize higher order information → notably, the second order gradient of the error function
  - Important because defines the “shape” of the error function
  - One example is the “Levenberg-Marquadt” (LM) algorithm:
    - Works by trying to find the zeros in the gradient space

    1. Using Newton's method:

    $$w_{n+1} = w_n - H^{-1} \frac{\partial E}{\partial w_n} \text{ , where } H_{ij} = \frac{\partial^2 E}{\partial w_i w_j}$$

    2. The matrix *H* is known as the “Hessian”; in the LM algorithm, this is approximated as:

    $$H_{ij} \approx \frac{\partial \epsilon}{\partial w_i} \cdot \frac{\partial \epsilon}{\partial w_j}$$

LM Algorithm (Cont'd)

3. In addition, a damping factor λ is introduced

$$H_{ij}^{damped} \approx \frac{\partial \epsilon}{\partial w_i} \cdot \frac{\partial \epsilon}{\partial w_j} + \lambda \delta(i-j)$$

- **The damping factor helps to lim it the scope of the update term**
  - When λ is small, we get the Newton Algorithm
  - When λ is big, the algorithm converges to the standard gradient descent algorithm
  - Hence, the choice of λ depends on the degree to which the linearity approximation is valid
- **In practice the following scheme can be used:**
  - Start with arbitrary value of λ, for e.g. 0.1
  - If, after one iteration, error decreases, reduce λ by factor of 10
  - If error increases, increase λ by factor of 10, revert to previous **w**
  - Repeat until error decreases.

Problem of overfitting

- **As before, two general approaches:**

i. Constraining the data

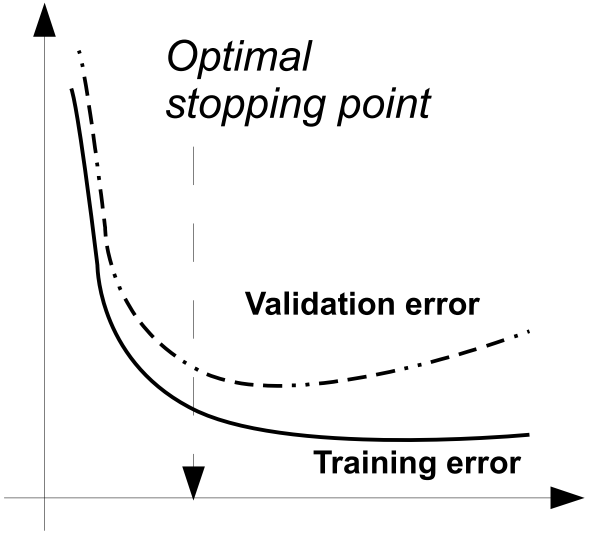
- Feature selection, Dimensionality reduction, etc..

ii. Constraining the model

- Similar to pruning → remove unnecesary complexity in the model..
- Will look at two techniques in particular:
  - Early Stopping
  - Optimal Brain Damage

Early stopping:

- Simple enough – stop training early!
- Motivation:
  - In training, NN typically fits the broad patterns in the data first
  - Once “low-hanging fruit” is picked, further training will tend to fit finer and finer levels of detail → overfitting!



Two potential solutions..

Optimal brain damage

- The idea is to discard "insignificant" weights (for e.g. by setting to zero)
- Consider following Taylor expansion:

$$\delta_i E = \underbrace{\frac{\partial E}{\partial w_i} w_i}_{\text{(i)}} + \underbrace{\frac{1}{2} \frac{\partial^2 E}{\partial w_i^2} w_i^2}_{\text{(ii)}} + \ldots$$

- Assuming that a local minimum has been reached, then term (i) is zero
- Use term (ii) to determine the "saliency" of a weight → i.e. the effect of removing one of the weights on the error
- The ***Optimal Brain Damage*** (OBD) algorithm exploits this:
  - Choose a large network architecture
  - Train until stopping criterion met
  - Compute second derivative for each weight, H_{ii} , and its corresponding saliency, $H_{ii}\, w_i^2$
  - Sort weights by saliency and remove a few low saliency weights
  - Repeat until termination criterion met

END OF COURSE!!

THANKS FOR LISTENING :-)

Appendix

(Effect of second order gradients on update term)

A simple example.. Case (a)

- **Consider the following cost function:**

$$E(x, y) = x^2 + y^2$$

  - Contour plot is as shown on right:
  - Error gradient is:

$$\nabla E(x, y) = [2\text{x}\, 2\text{y}]^T$$

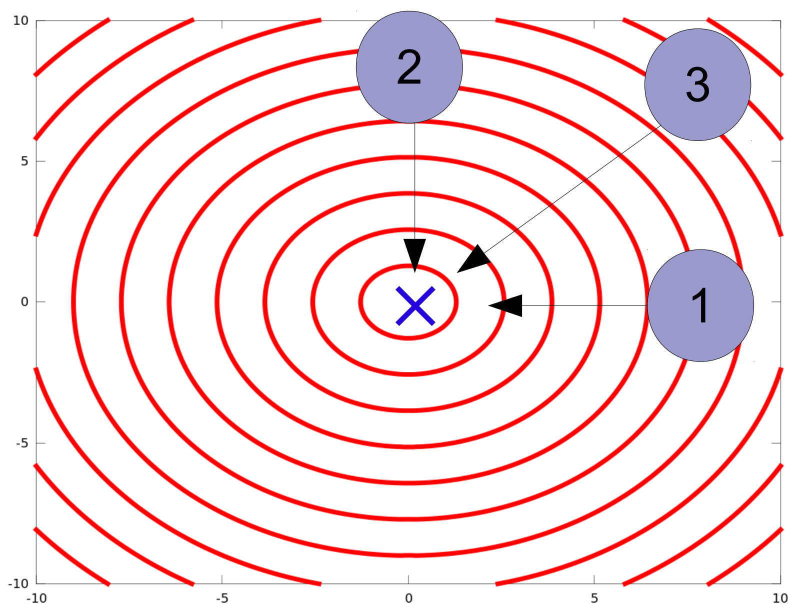
  - Let's evaluate this at a number of points:

(1) $\nabla E(1,0) = [2 \quad 0]^T$

(2) $\nabla E(0,1) = [0 \quad 2]^T$

(3) $\nabla E(1,1) = [2 \quad 2]^T$

- **Directions of the negative vector gradient terms are shown on right**
  - ('X' marks the spot!)
  - Hence, for this example, all vectors point to the global minimum



Case (b)

- **Next, we modify the cost function as follows:**

$$E(x,y)=5x^2+y^2$$

  - Error gradient is:

$$\nabla E(x,y)=[10x \quad 2y]^T$$

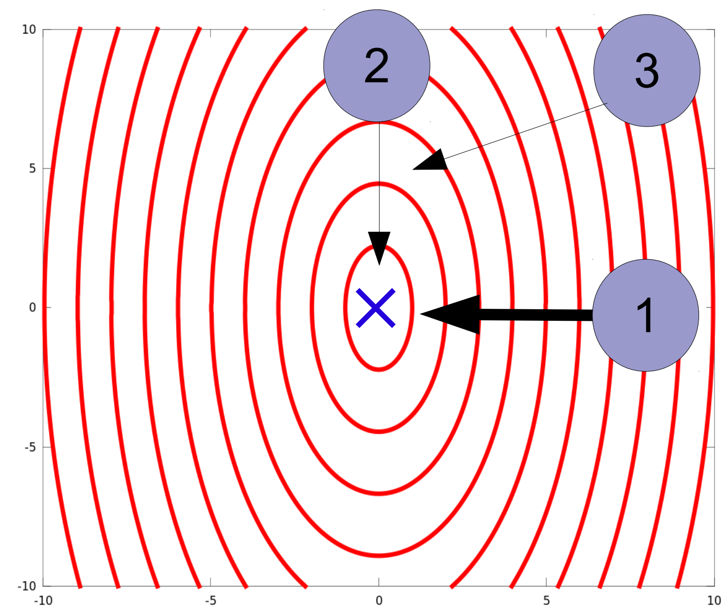
  - Let's evaluate this at a number of points:

(1) $$\nabla E(1,0)=[10 \quad 0]^T$$

(2) $$\nabla E(0,1)=[0 \quad 2]^T$$

(3) $$\nabla E(1,1)=[10 \quad 2]^T$$

- **Directions of the negative vector gradient terms are shown on right**
  - Gradients at ***(1)*** and ***(2)*** point in the correct directions, but..
  - … magnitudes are not proportional to distance from minima!
  - Gradient at ***(3)*** points in the wrong direction (slow convergence)



Case (c)

- **Next, we modify the cost function as follows:**

$$E(x,y)=x^2+y^2+xy$$

  - Error gradient is:

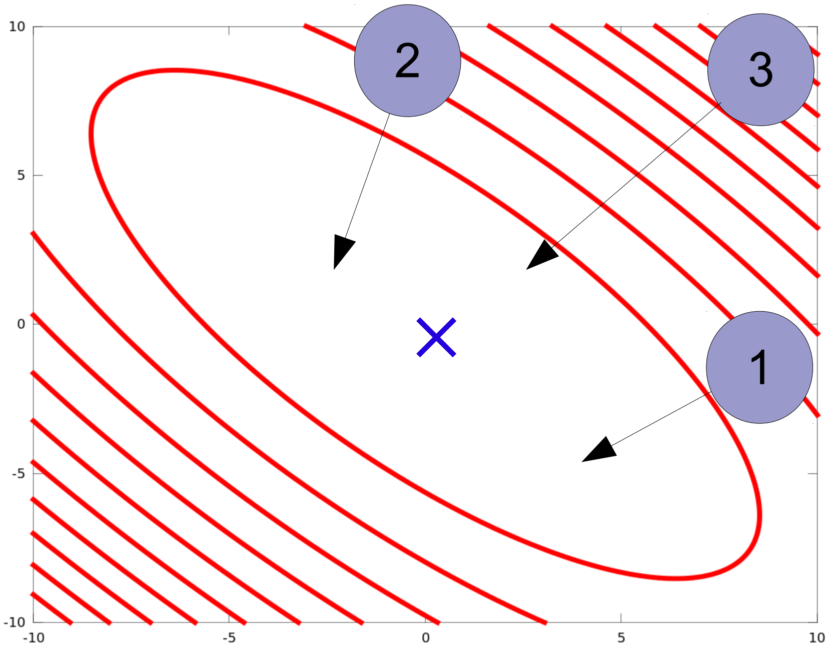
$$\nabla E(x,y)=[2x+y \quad 2y+x]^T$$

  - Let's evaluate this at a number of points:

(1) $\nabla E(1,0)=[2 \quad 1]^T$

(2) $\nabla E(0,1)=[1 \quad 2]^T$

(3) $\nabla E(1,1)=[3 \quad 3]^T$



- **Directions of the negative vector gradient terms are shown on right**
  - For this example, we can see that the negative gradient at ***(3)*** still points at the global optimum
  - Gradients at ***(1)*** and ***(2)*** are "off"

Masdar INSTITUTE

Use of second order gradients

- **For cases (b) and (c), let's now investigate effect of using the Hessian..**
  - For bivariate function (in *x* and *y*), Hessian is given by (1) $\rightarrow$

$$H(e)=\begin{bmatrix} \frac{\partial^2 E}{\partial x^2} & \frac{\partial^2 E}{\partial x \partial y} \\ \frac{\partial^2 E}{\partial y \partial x} & \frac{\partial^2 E}{\partial y^2} \end{bmatrix} \quad \ldots (1)$$

  - For case (b):

$$E(x,y)=5x^2+y^2 \Rightarrow H(e)=\begin{bmatrix} 10 & 0 \\ 0 & 2 \end{bmatrix}$$

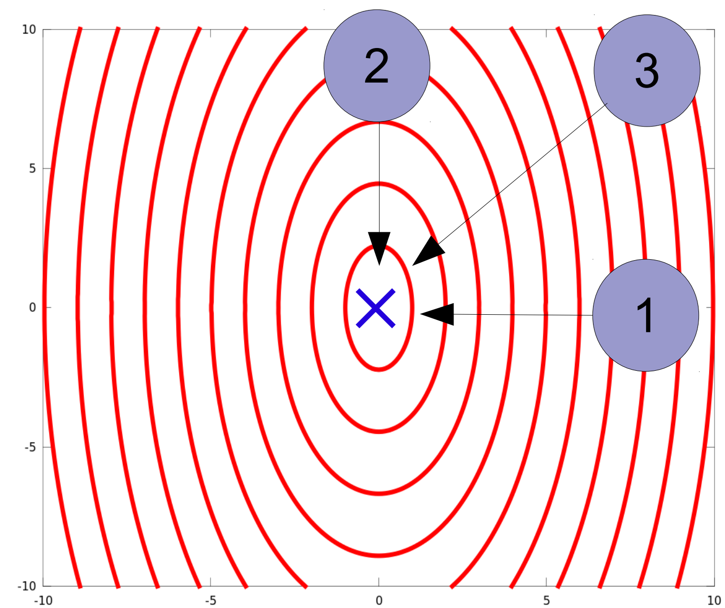
$$\Rightarrow H(e)^{-1}=\begin{bmatrix} 0.1 & 0 \\ 0 & 0.5 \end{bmatrix}$$

  - Let's evaluate this at points 1, 2 and 3:

$$H(e)^{-1}.\nabla E(1,0)=\begin{bmatrix} 0.1 & 0 \\ 0 & 0.5 \end{bmatrix}.\begin{bmatrix} 10 \\ 0 \end{bmatrix}=\begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

$$H(e)^{-1}.\nabla E(0,1)=\begin{bmatrix} 0.1 & 0 \\ 0 & 0.5 \end{bmatrix}.\begin{bmatrix} 0 \\ 2 \end{bmatrix}=\begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

$$H(e)^{-1}.\nabla E(1,1)=\begin{bmatrix} 0.1 & 0 \\ 0 & 0.5 \end{bmatrix}.\begin{bmatrix} 10 \\ 2 \end{bmatrix}=\begin{bmatrix} 1 \\ 1 \end{bmatrix}$$



- **Update terms now point exactly where we need em ::**

Masdar INSTITUTE

Cont'd

- For case (c):

$$E(x,y)=x^2+y^2+xy \Rightarrow H(e)=\begin{bmatrix}2 & 1\\ 1 & 2\end{bmatrix}$$

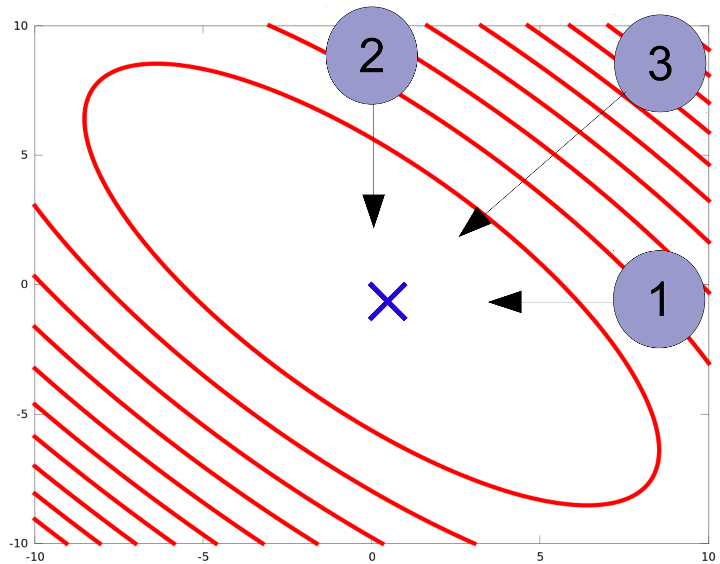
$$\Rightarrow H(e)^{-1}=\begin{bmatrix}0.667 & -0.333\\ -0.333 & 0.667\end{bmatrix}$$

- Let's evaluate this at points 1, 2 and 3:

$$H(e)^{-1}.\nabla E(1,0)=\begin{bmatrix}0.667 & -0.333\\ -0.333 & 0.667\end{bmatrix}.\begin{bmatrix}2\\ 1\end{bmatrix}=\begin{bmatrix}1\\ 0\end{bmatrix}$$

$$H(e)^{-1}.\nabla E(0,1)=\begin{bmatrix}0.667 & -0.333\\ -0.333 & 0.667\end{bmatrix}.\begin{bmatrix}1\\ 2\end{bmatrix}=\begin{bmatrix}0\\ 1\end{bmatrix}$$

$$H(e)^{-1}.\nabla E(1,1)=\begin{bmatrix}0.667 & -0.333\\ -0.333 & 0.667\end{bmatrix}.\begin{bmatrix}3\\ 3\end{bmatrix}=\begin{bmatrix}1\\ 1\end{bmatrix}$$



- **Again, update terms now point exactly where we need em'!!**