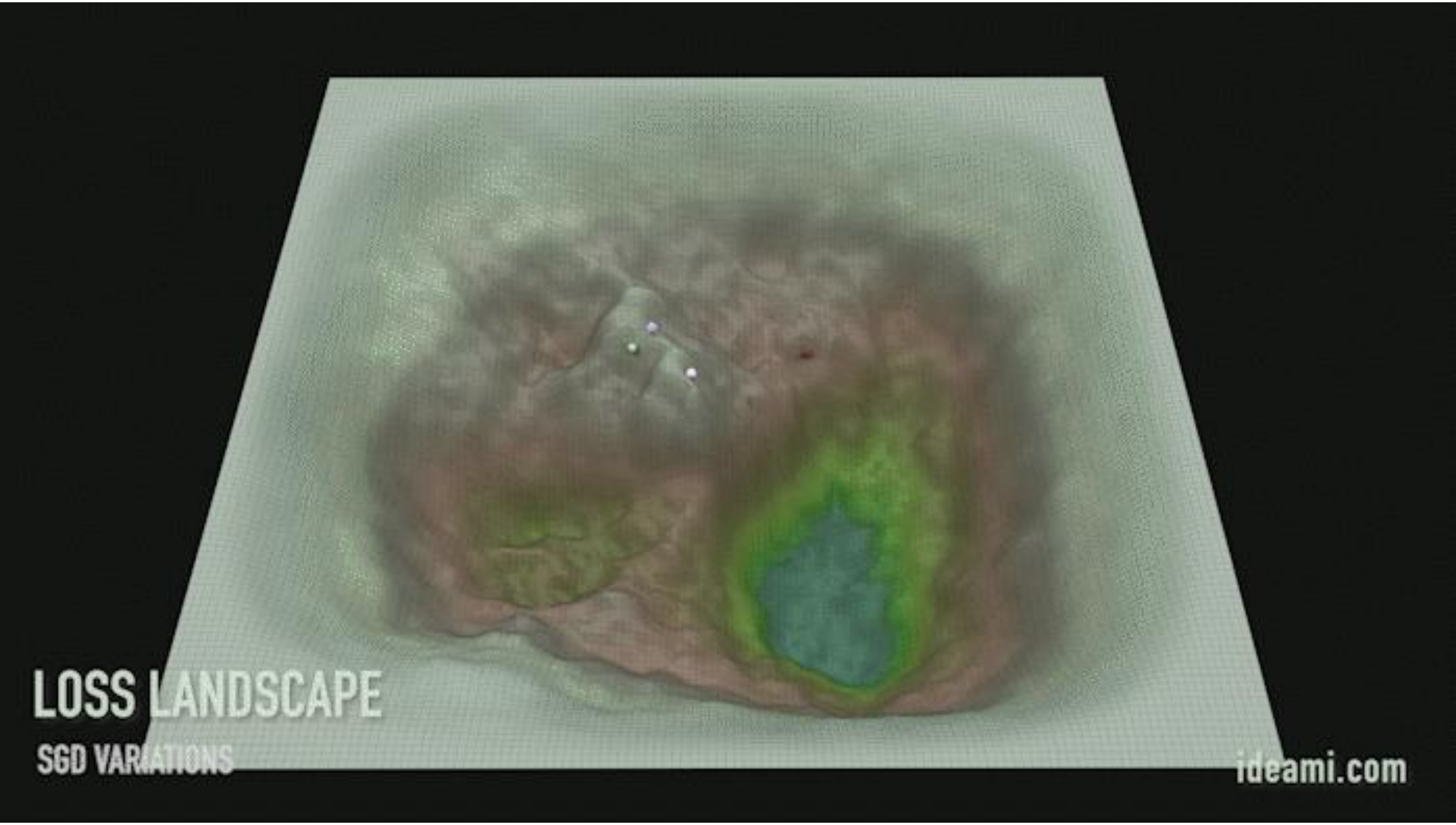
Gradient Descent

Erisa Terolli

Intuition

Intuition

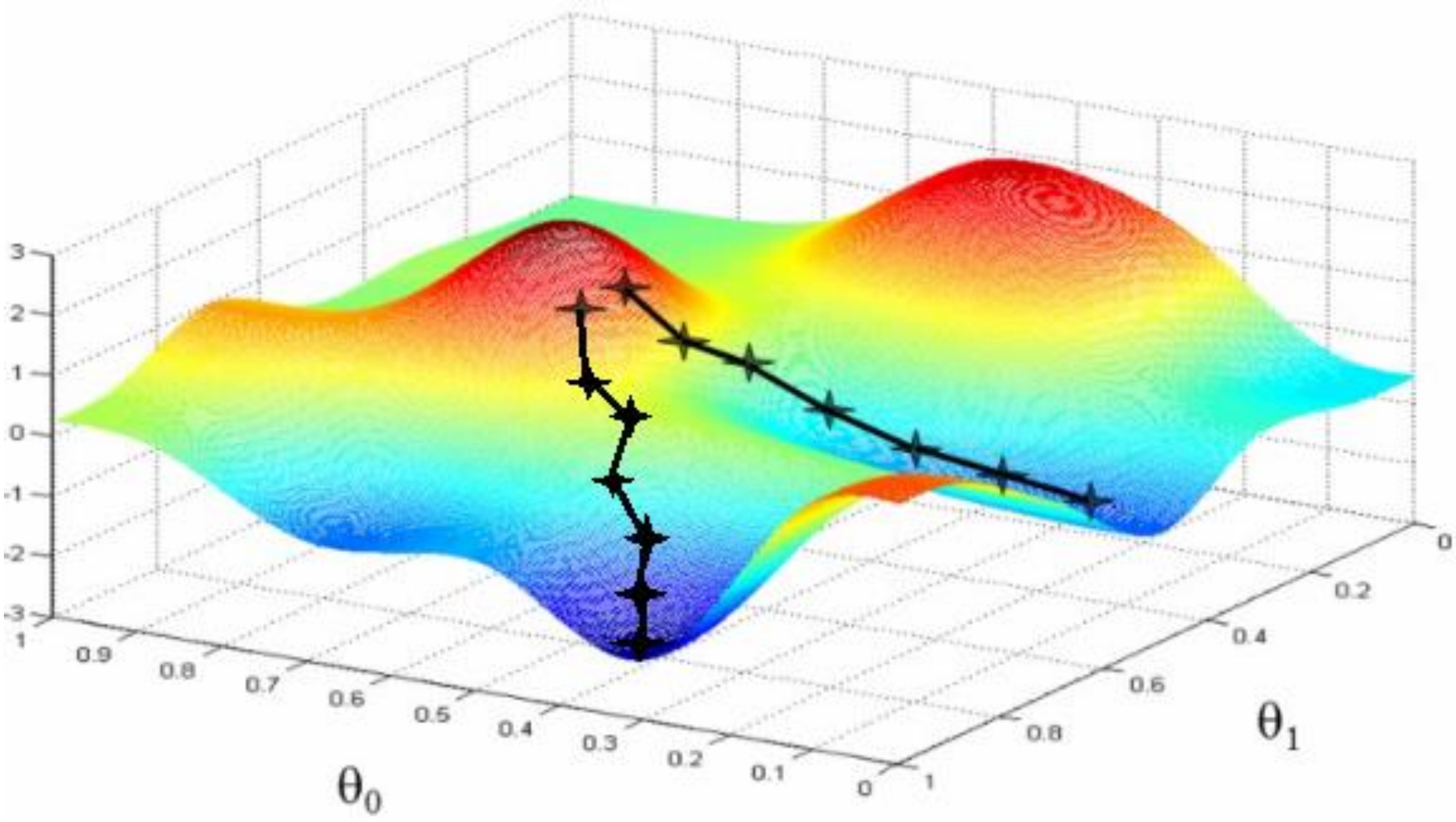


Motivation

- Prof. Konrad Kording, Penn University, twitted Dec 1
  - “I think that the brain almost certainly approximates gradient descent. And here is why:
    - Any learning episode only appears to change the brain a tiny bit
    - Given that the brain appears to be quite noisy and somewhat linear, this means that we can almost certainly locally approximate the task loss L linearly around the starting parameters W_0 as $L(W\_0+\Delta W) \approx L(W\_0)+\Delta W \, \nabla L(W)$
    - This immediately implies that the brain can only get better if the changes induced by a learning episode are proportional to the loss gradient $\nabla L(W)$. Animals almost always improve behaviorally.
    - I also assume this approximation to be good. For a given improvement, the gradient descent solution of $\Delta W=-\gamma \nabla L$ is the one that would change the brain least. The size of the overall change to the brain is a central factor in across-task interference. Minimal interference!
    - This logic is why I have trouble seeing gradient descent as "just another theory that may be right or wrong" - it has a strong normative justification and, under certain rather harmless assumptions, must be at least correlated to plasticity in the brain.”
- Prof. Yann LeCun, NYU and Facebook, twitted Dec 4
  - “Another reason why the brain might be using gradient-based learning is that the known alternatives to it are too inefficient to be usable at the scale of a brain.”

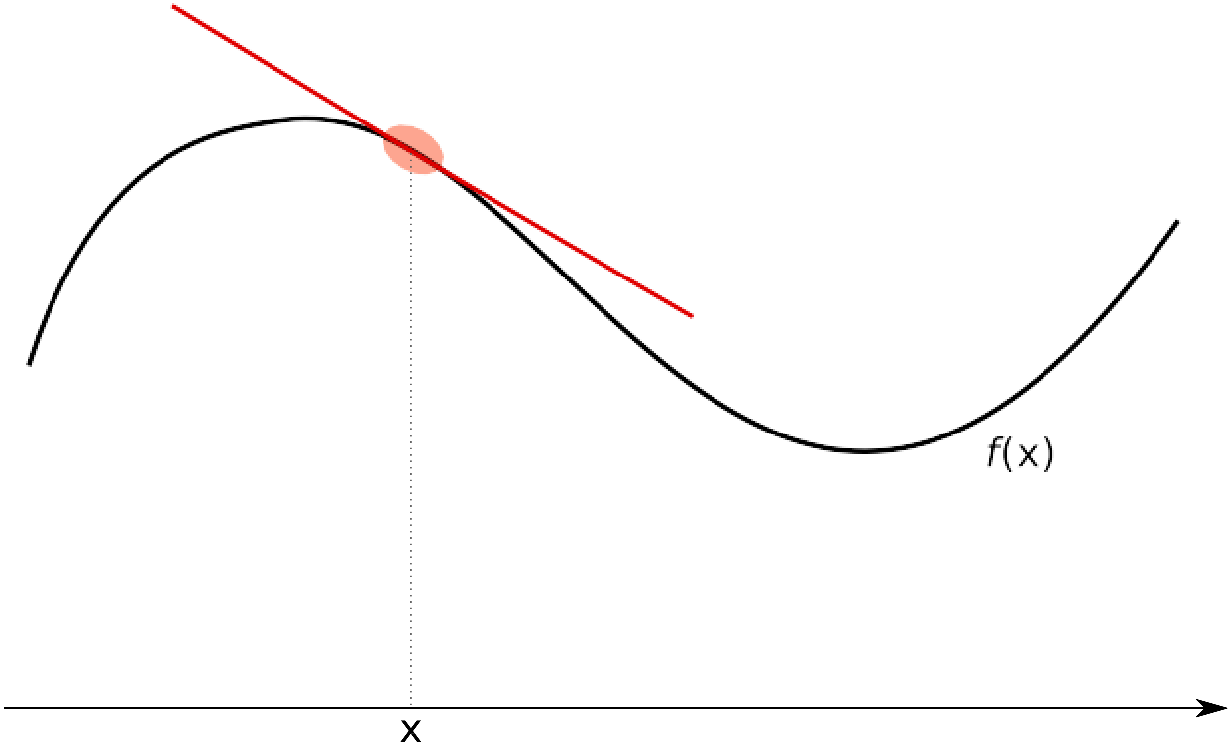
Task

- Find the parameters $\boldsymbol{\theta}$ that minimize the function $f(\theta)$



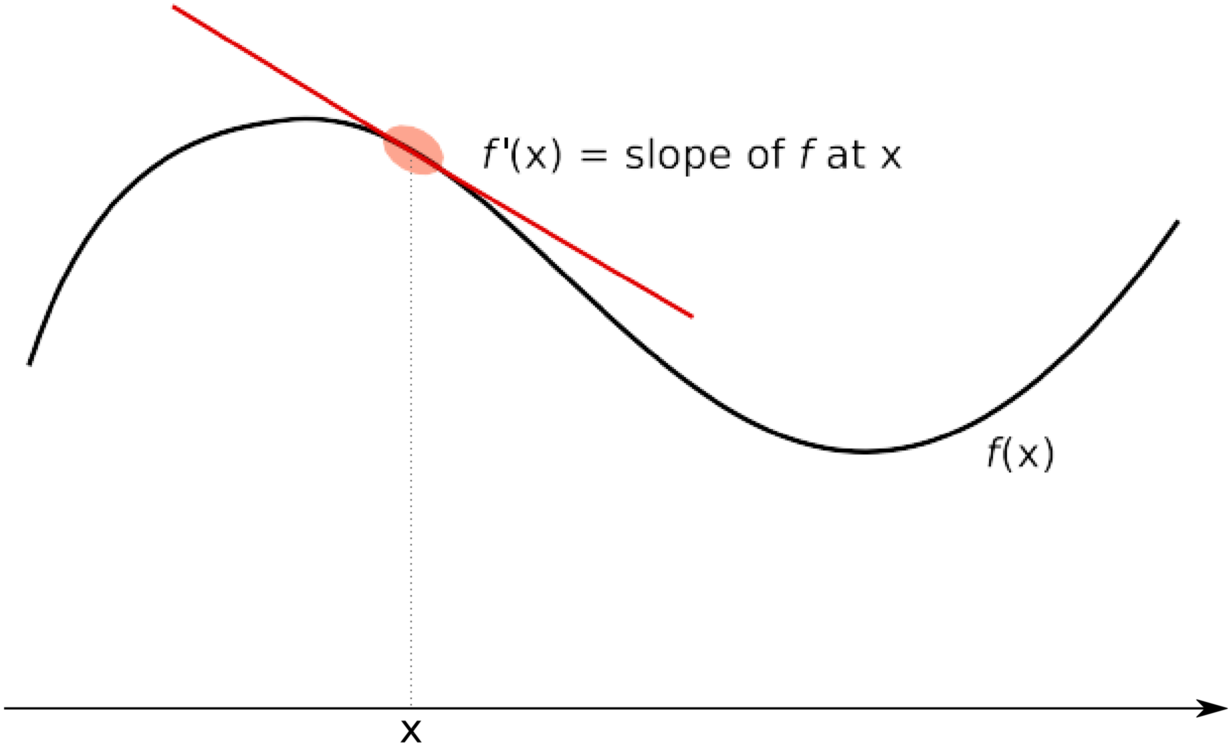
Task in 1D

- Find the x that minimizes the $f(x)$



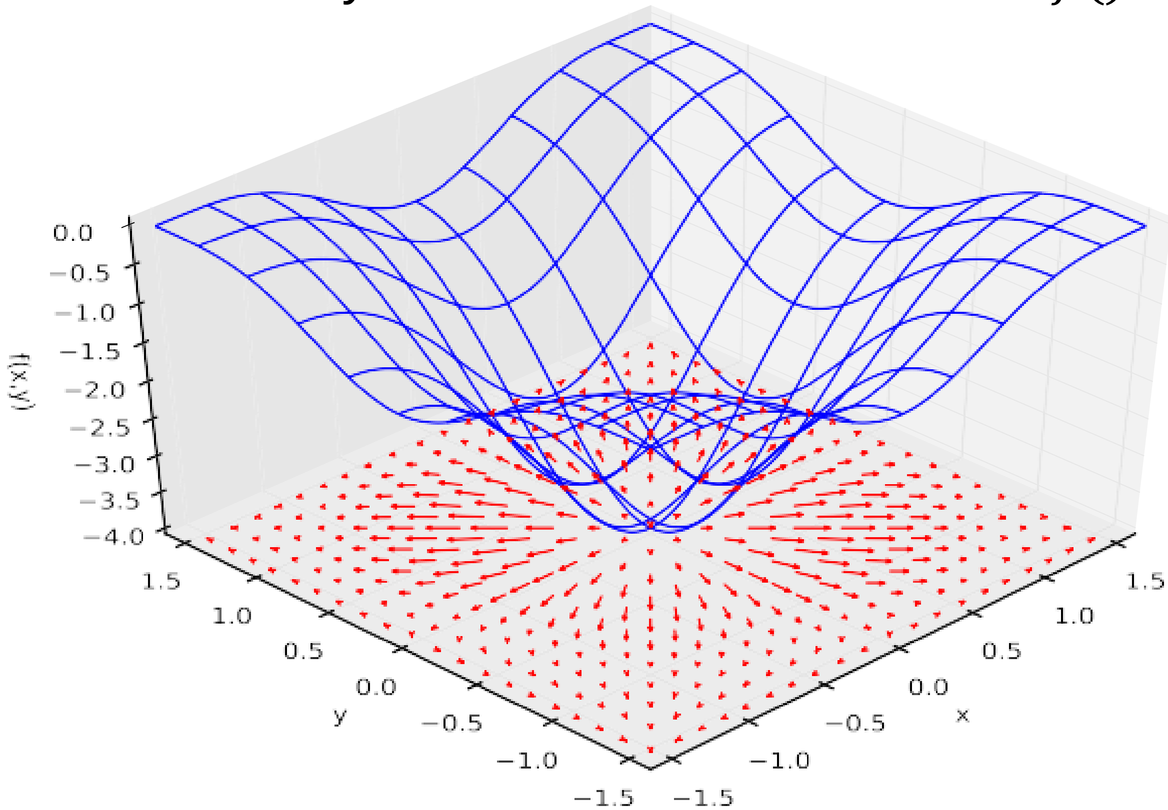
Gradient of a function in 1D

- The derivative *f'*(*x*) of *f*(*x*) tells the direction and intensity of the increase of *f*() at *x*



Gradient of a function

- The gradient $\nabla f(x_1, x_2, \dots, x_d)$ of $f(x_1, x_2, \dots, x_d)$ tells the direction and intensity of the maximum increase of $f()$ at $(x_1, x_2, \dots, x_d)$

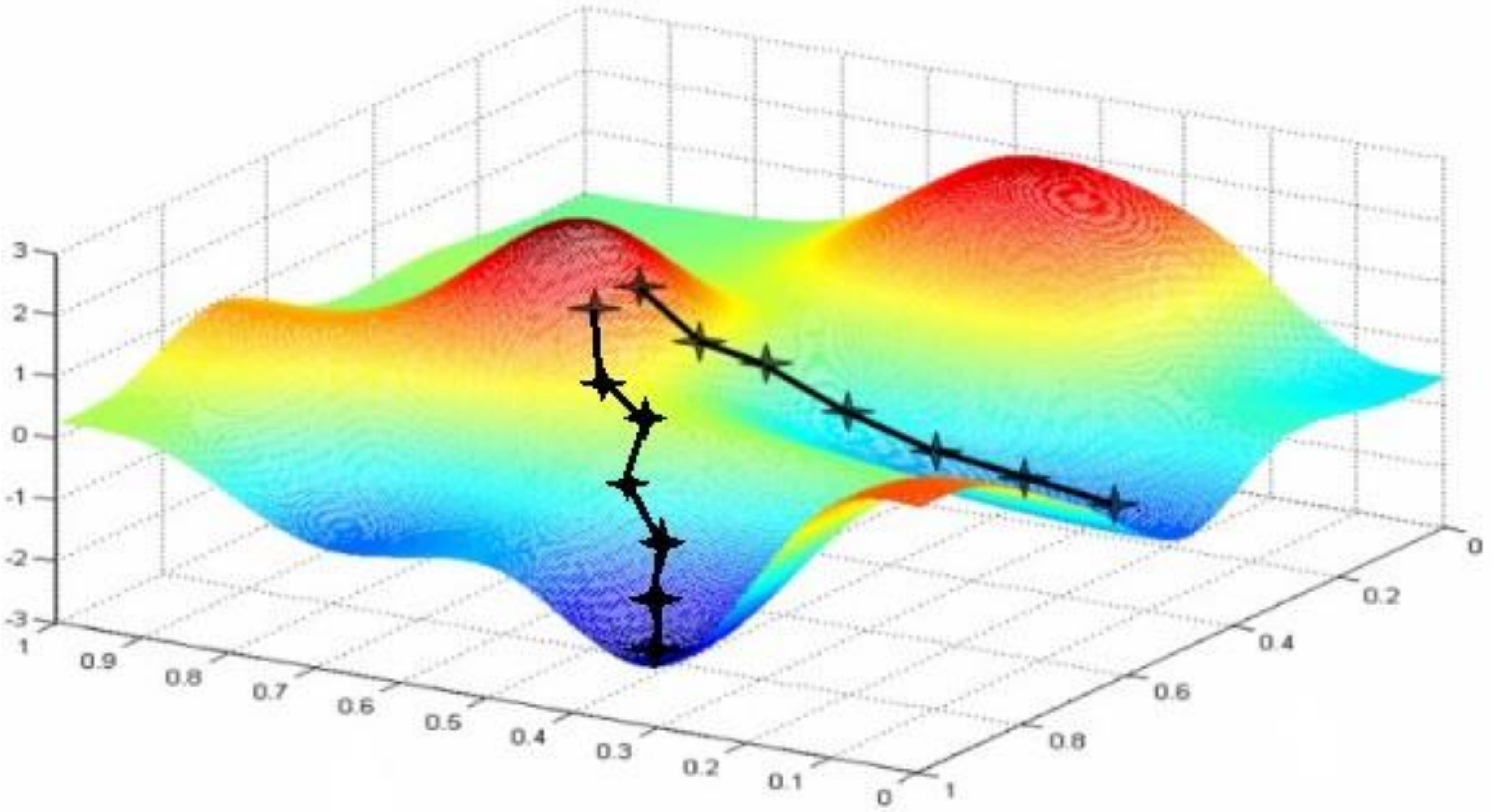


$f(x_1, x_2, \dots, x_d)$

$f(x_1, x_2, \dots, x_d)$

Gradient Descent

- Find $\min f()$ by repeatedly following $-\nabla f()$



Gradient Descent, more formally (1/2)

- Definition of gradient

$$\nabla f = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \\ \vdots \\ \frac{\partial f}{\partial x_d} \end{bmatrix}$$

examples:

$$f(\mathbf{x}) = 3x_1 - 7x_2 \text{ has gradient } \nabla f = \begin{bmatrix} 3 \\ -7 \end{bmatrix}$$

$$f(\mathbf{x}) = 3x_1^2 - 7 \text{ has gradient } \nabla f = \begin{bmatrix} 6x_1 \\ 0 \end{bmatrix}$$

Gradient Descent, more formally (2/2)

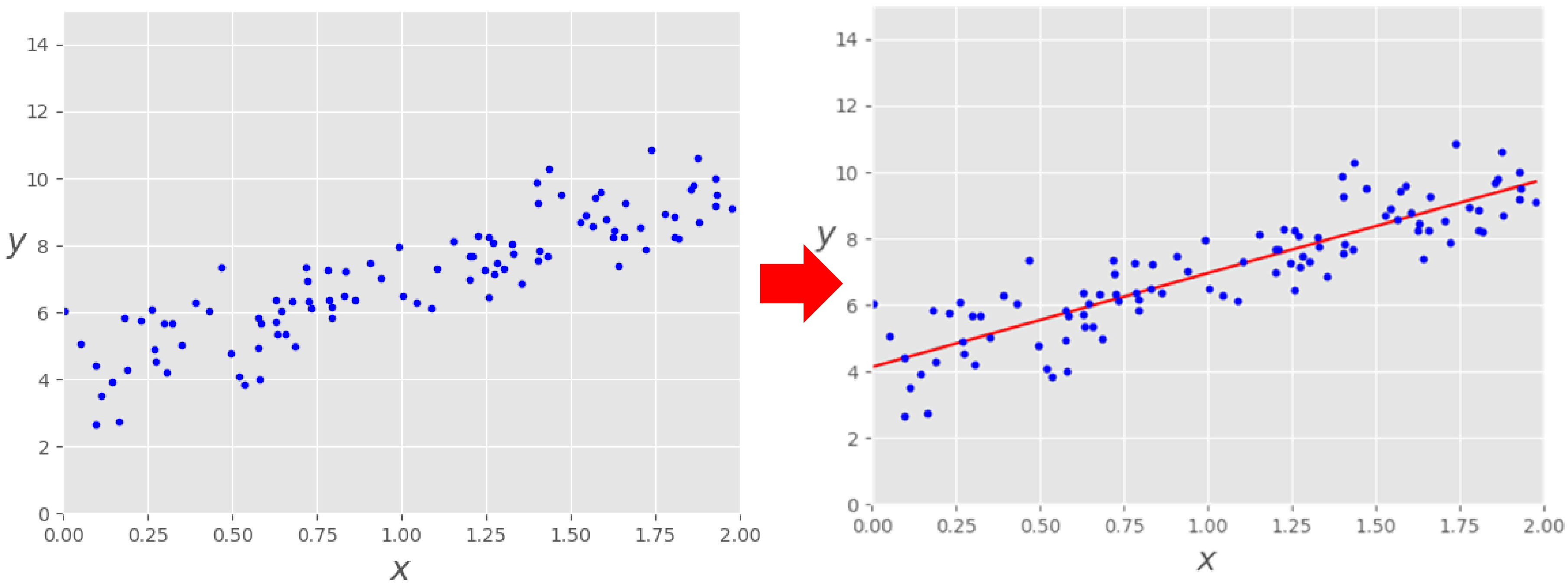
- Pseudocode
  - pick an arbitrary starting point $\mathbf{x^0}$
  - repeat until convergence:

$$\mathbf{x^{i+1}} = \mathbf{x^i} - \alpha \nabla f(\mathbf{x^i})$$

- the non-negative parameter α is called *learning rate*

Application to Linear Regression

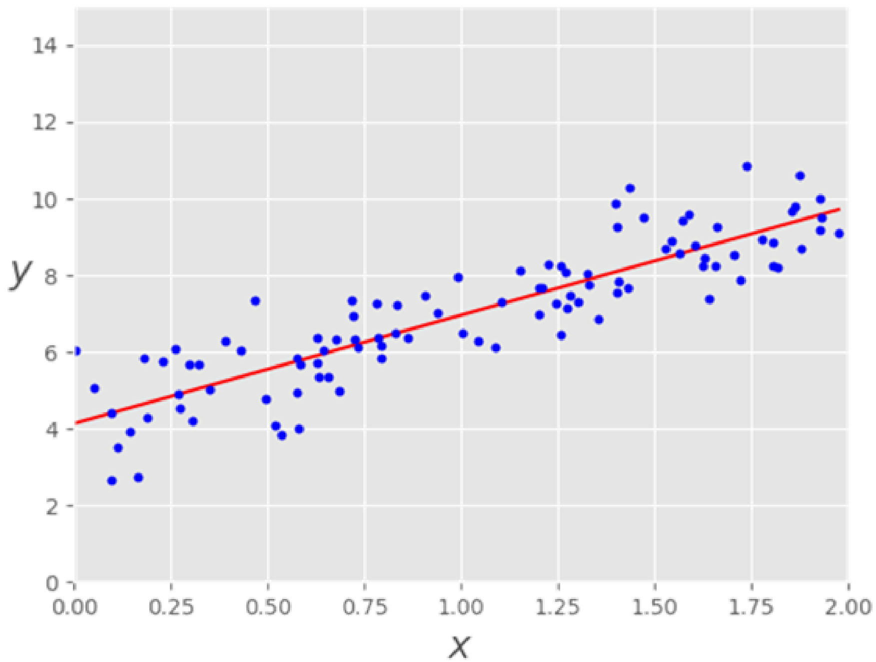
- Find the hypothesis function which minimize the loss
  - $J(\theta) = \frac{1}{2m}\sum_{k=1}^{m}\left(\left(h^{(k)}(\theta)\right) - y^{(k)}\right)^2$
  - In this case: $h^{(k)}(\theta) = \theta_0 + \theta_1 x^{(k)} = \left[1\ x^{(k)}\right] \cdot \boldsymbol{\theta} = \boldsymbol{a}^{(k)} \cdot \boldsymbol{\theta}$



Application to Linear Regression

- Find the hypothesis function which minimize the loss
  - $J(\theta) = \frac{1}{2m}\sum_{k=1}^{m}(\left(h^{(k)}(\theta)\right) - y^{(k)})^2$
  - In this case: $h^{(k)}(\theta) = \theta_0 + \theta_1 x^{(k)} = \left[1\ x^{(k)}\right] \cdot \boldsymbol{\theta} = \boldsymbol{a}^{(k)} \cdot \boldsymbol{\theta}$

- Analytical solution
  - $A = [\mathbf{1}\ \boldsymbol{x}]$
  - $\boldsymbol{y} = A\,\boldsymbol{\theta}$
  - $\theta = (A^T A)^{-1} A^T \boldsymbol{y}$



Gradient descent for linear regression

- Find the hypothesis function which minimize the loss
  - $J(\theta) = \frac{1}{2m}\sum_{k=1}^{m}(\left(h^{(k)}(\theta)\right) - y^{(k)})^2$
  - In this case: $h^{(k)}(\theta) = \theta_0 + \theta_1 x^{(k)} = \left[1\ x^{(k)}\right] \cdot \boldsymbol{\theta} = \boldsymbol{a}^{(k)} \cdot \boldsymbol{\theta}$

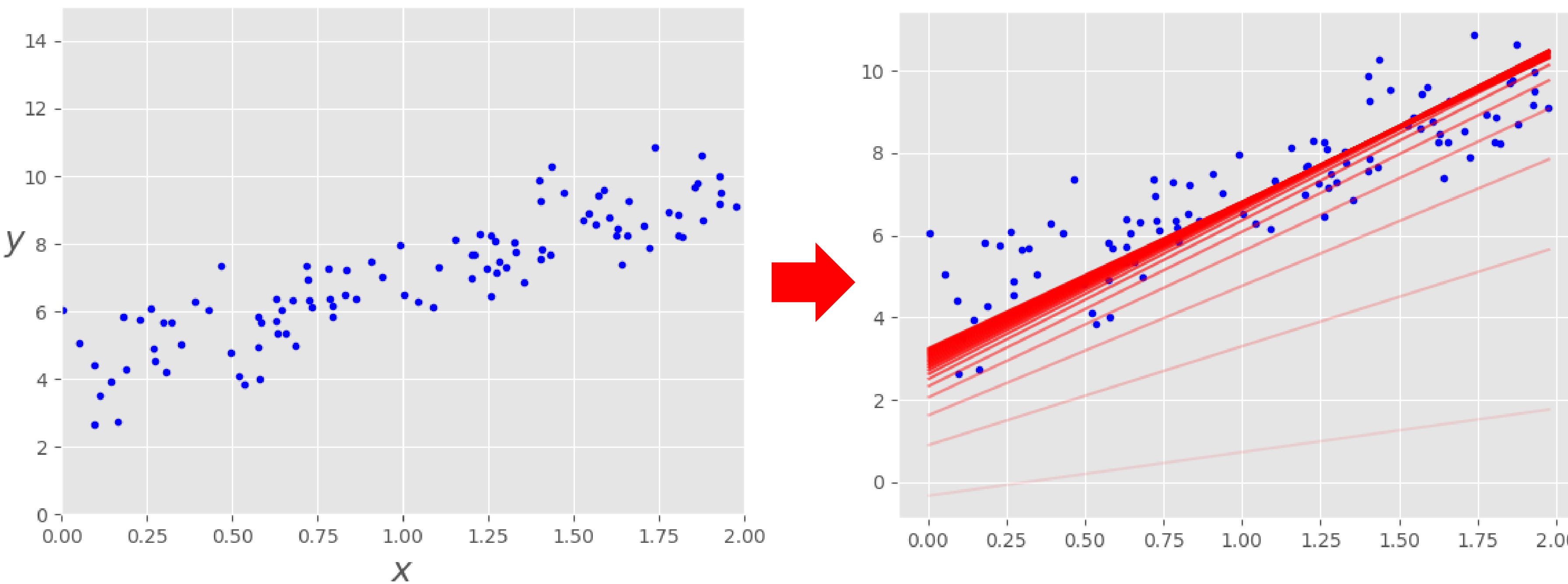


Illustration of the hypothesis (θ_0, θ_1) space

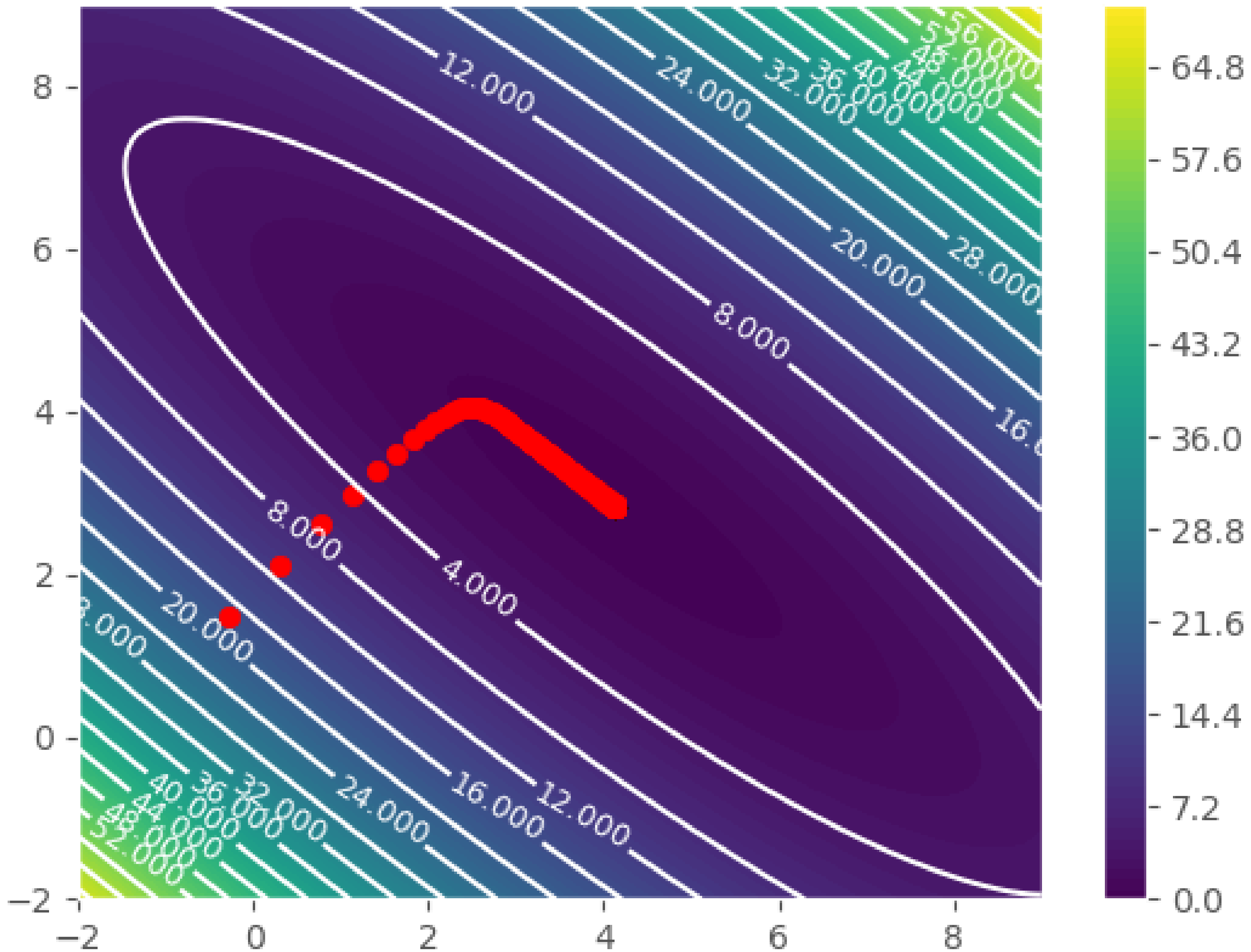
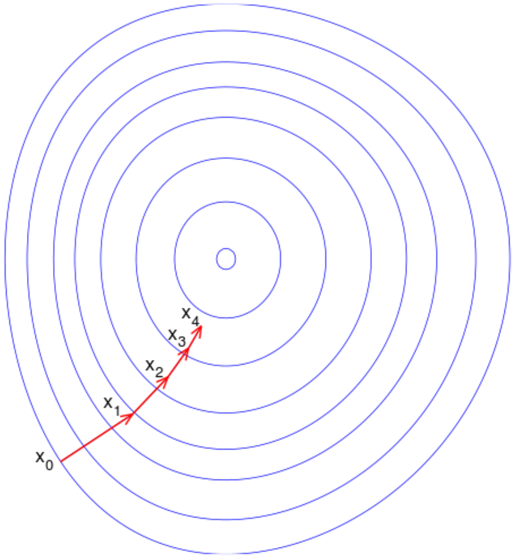


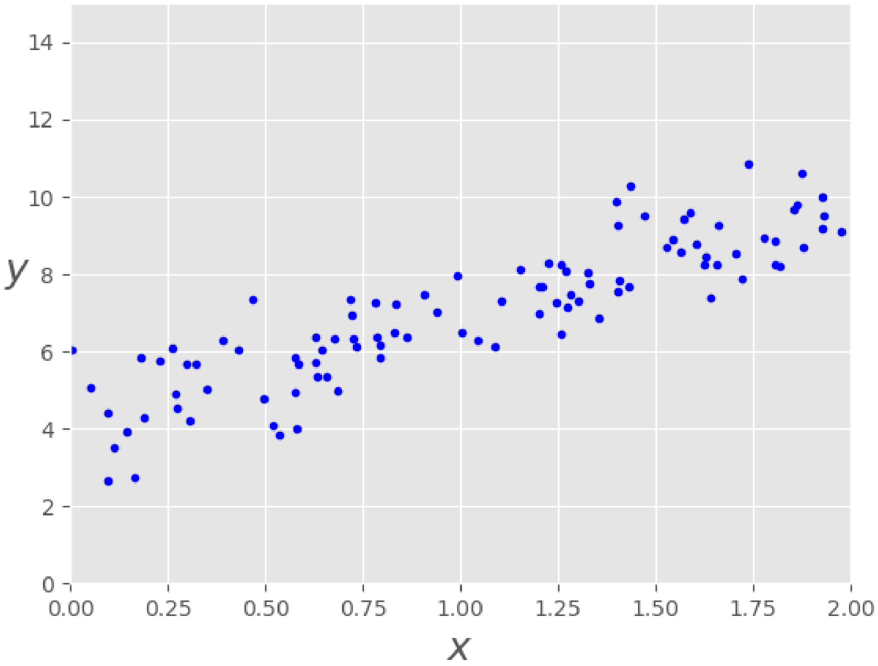
Illustration of Gradient descent

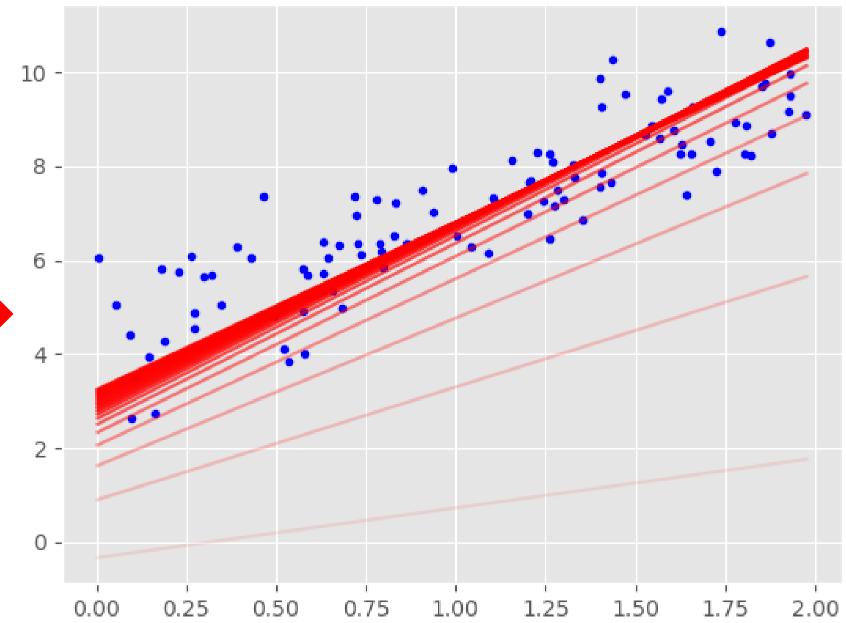
- $-\nabla J(\boldsymbol{\theta}^{(i)})$ is a descent direction
- $\boldsymbol{\theta}^{(i+1)} = \boldsymbol{\theta}^{(i)} - \alpha \nabla J(\boldsymbol{\theta}^{(i)})$
  - $\boldsymbol{\Delta\theta}^{(i)}$ is the step, or search direction
  - α is the step size, or step length, or learning rate
    - Too small α will cause slow convergence
    - Too large α could cause overshoot the minima and diverge



Gradient descent for linear regression

- Find the hypothesis function which minimize the loss
  - $J(\theta) = \frac{1}{2m}\sum_{k=1}^{m}((h^{(k)}(\theta)) - y^{(k)})^2$
  - In this case: $h^{(k)}(\theta) = \theta_0 + \theta_1 x^{(k)} = [1\ x^{(k)}] \cdot \boldsymbol{\theta} = \boldsymbol{a}^{(k)} \cdot \boldsymbol{\theta}$





Gradient Descent
One data sample

- Gradient of a square error function
  - $J(\boldsymbol{\theta}) = \frac{1}{2}((h\,(\boldsymbol{\theta})) - y)^2$
    - with: $h(\boldsymbol{\theta}) = \theta_0 + \theta_1 x = [1\ x] \cdot \boldsymbol{\theta} = \boldsymbol{a} \cdot \boldsymbol{\theta}$
  - $\frac{\delta J(\boldsymbol{\theta})}{\delta \boldsymbol{\theta}} = \left(\left(h\,(\boldsymbol{\theta})\right) - y\right) \frac{\delta h(\boldsymbol{\theta})}{\delta \boldsymbol{\theta}} = (\boldsymbol{a}\,\boldsymbol{\theta} - y)\,\boldsymbol{a}$
  - Or equivalently
  - $\frac{\delta J(\boldsymbol{\theta})}{\delta \theta_j} = \left(\left(h\,(\boldsymbol{\theta})\right) - y\right) \frac{\delta h(\boldsymbol{\theta})}{\delta \theta_j} = (\boldsymbol{a}\,\boldsymbol{\theta} - y) a_j$

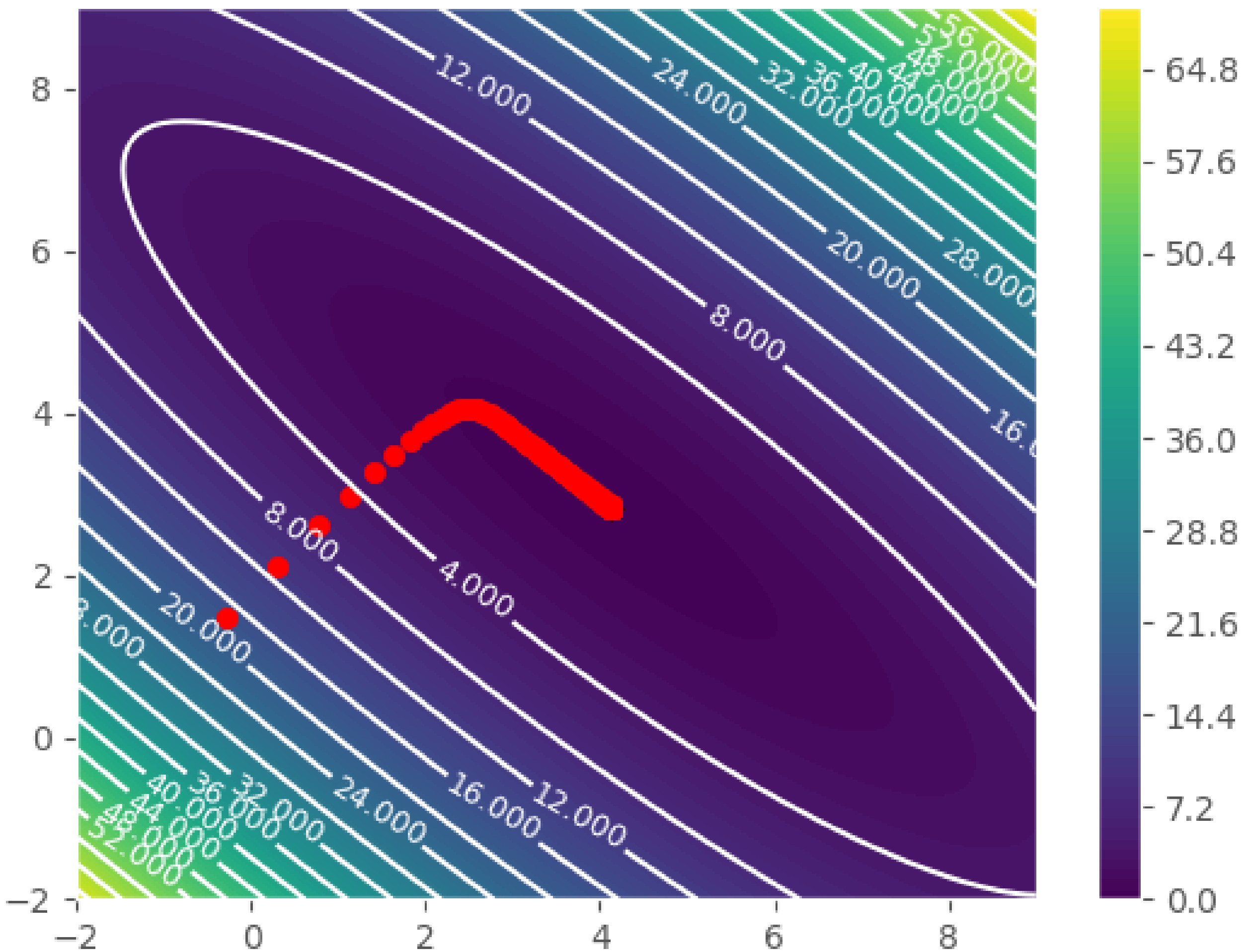
Gradient Descent
m data points

- Pseudocode
  - pick an arbitrary starting point $\boldsymbol{\theta}^{(0)}$
  - repeat until convergence:
    - $\boldsymbol{\theta}^{(i+1)} = \boldsymbol{\theta}^{(i)} - \alpha \nabla J(\boldsymbol{\theta}^{(i)})$
- the non-negative parameter *α* is called *learning rate*

- Inside the loop
  - $\theta_j^{(i+1)} = \theta_j^{(i)} - \alpha \frac{\delta J(\boldsymbol{\theta})}{\delta \theta_j}$ simultaneously for every *j* = *1, … , d*

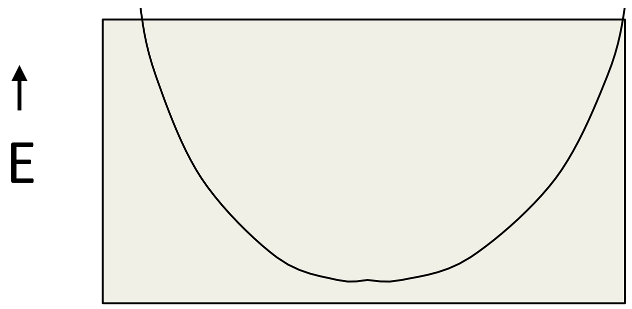
- Using *m* data points ($\boldsymbol{x}^{(1)}$, $y^{(1)}$), … , ($\boldsymbol{x}^{(m)}$, $y^{(m)}$)
  - $\theta_j^{(i+1)} = \theta_j^{(i)} - \alpha \left(\frac{1}{m} \sum_{k=1}^{m} \left(\boldsymbol{a}^{(k)} \boldsymbol{\theta}^{(i)} - y^{(k)} \right) a_j^{(k)} \right)$

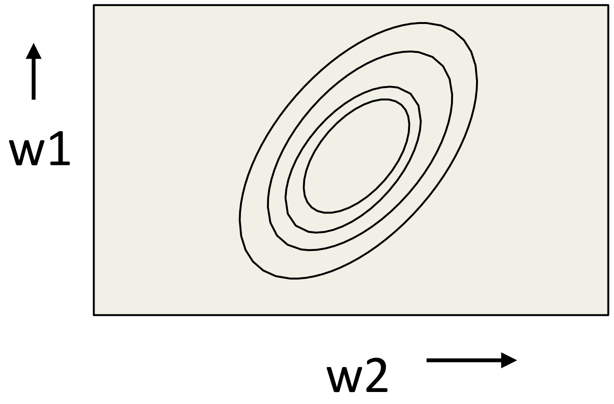
Illustration of the hypothesis (θ_0, θ_1) space



The error surface for a linear neuron

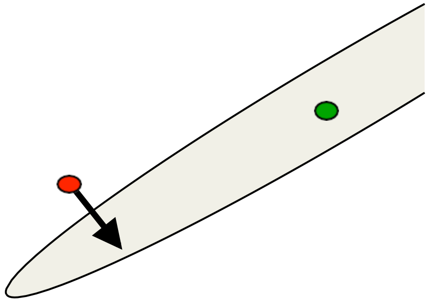
- The error surface for a linear neuron with a squared error is a quadratic bowl
  - → Use gradient descent to learn *W*
- For multi-layer non-linear nets the error surface is more complex
  - But locally, a piece of a quadratic bowl is usually a very good approximation





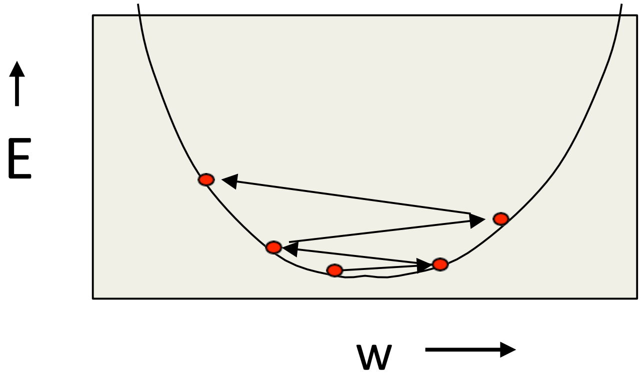
Convergence speed

- Going downhill reduces the error, but the direction of steepest descent does not point at the minimum unless the ellipse is a circle
  - The gradient is big in the direction in which we only want to travel a small distance
  - The gradient is small in the direction in which we want to travel a large distance
- Even for non-linear multi-layer nets the error surface is locally quadratic, so the same speed issues apply



How the learning goes wrong

- If the learning rate is big, the weights slosh to and from across the ravine
  - If the learning rate is too big, this oscillation diverges.
- What we would like to achieve
  - Move quickly in directions with small but consistent gradients
  - Move slowly in directions with big but inconsistent gradients



Python demo

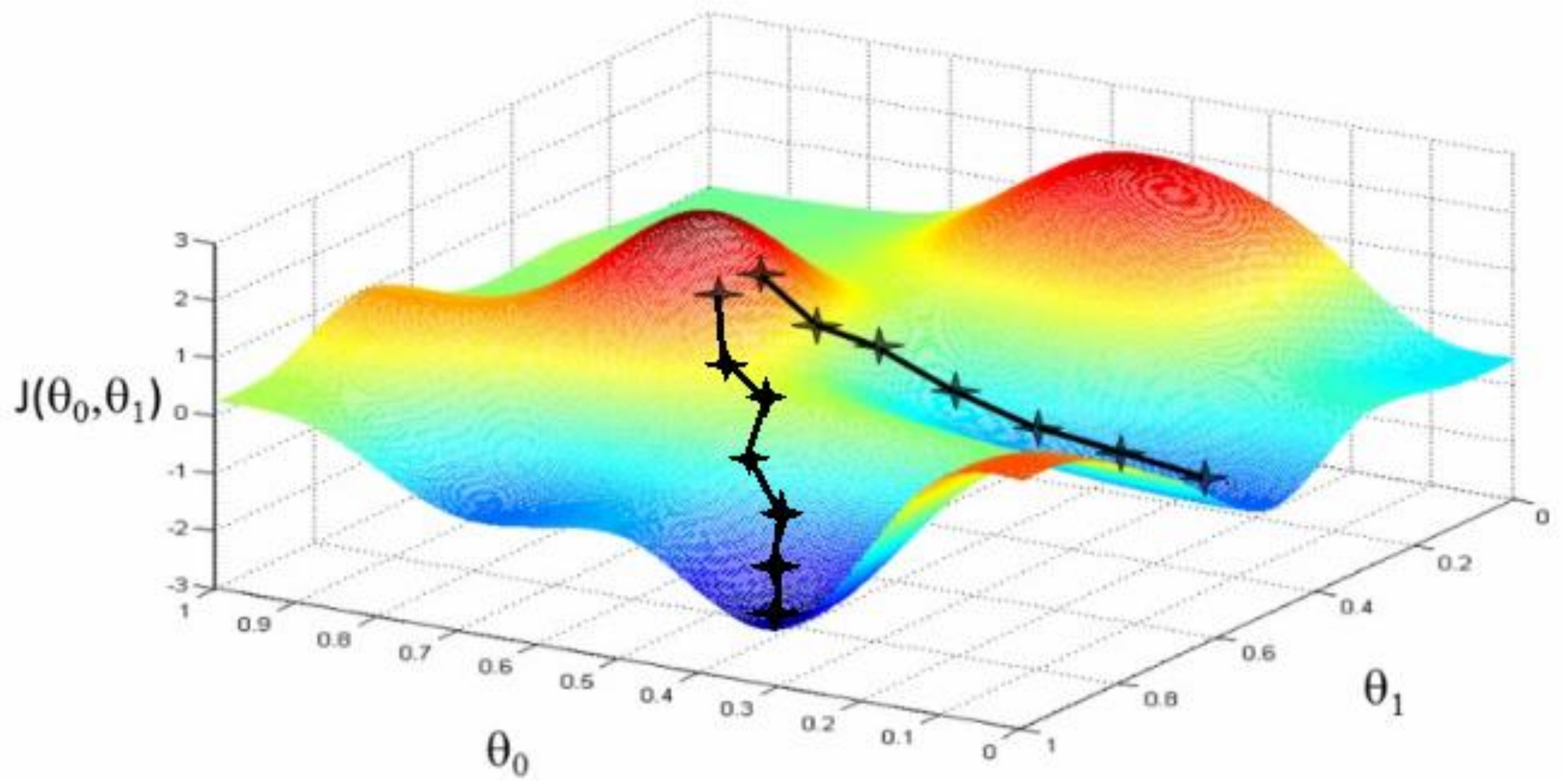
- 1;
return c; }
push(a[c]); }
return
/gm, " "), b
inp_array.length;
for (
(c.push(inp_array[a]),
inp_array));
-1 < b
-1 < b && a.
function use_array(a, b) {
for (
(a, b) {
(var c = -1, d

Pros and Cons

- Pros
  - Can be applied to every dimension and space (even possible to infinite dimension)
  - Easy to implement
- Cons
  - Local minima problem
  - Relatively slow close to minimum
  - For non-differentiable functions, gradient methods are ill-defined

Local minima in Gradient Descent

- Find the parameters (θ_0, θ_1) that minimize the cost $\mathrm{J}(\theta_0, \theta_1)$



Types of Gradient Descent

- According to the type of data ingestion
  - Stochastic Gradient Descent Algorithm
  - Batch Gradient Descent Algorithm
- Based on differentiation techniques
  - First order Differentiation
  - Second order Differentiation

Stochastic gradient descent

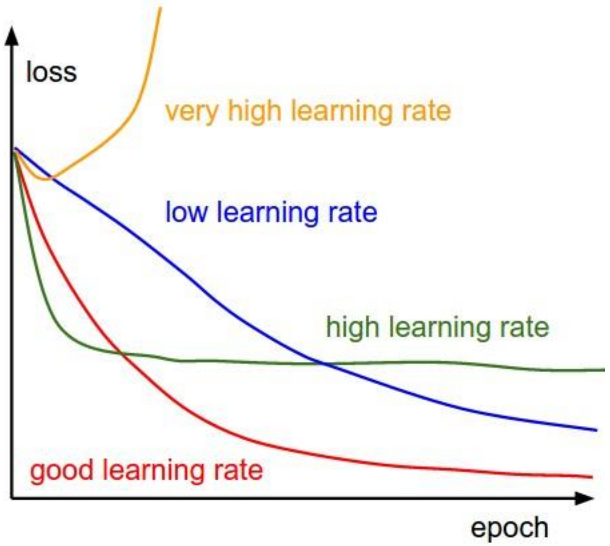
- For large training sets, the evaluation of the gradient over all samples may be expensive
- **Stochastic** or “**online**” gradient descent approximates the true gradient by a gradient at a single example
  - Pseudocode:
  - Choose an initial vector of parameters θ_0 and learning rate α
  - Repeat until convergence:
    - Randomly shuffle examples in the training set
    - For $k = 1,2, \dots m$, do:
      - $\boldsymbol{\theta}^{(i+1)} = \boldsymbol{\theta}^{(i)} - \alpha \, \nabla J^{(k)}(\boldsymbol{\theta}^{(i)})$
- Normally preferable: **batch** gradient descent
  - Consider a mini-batch at each step
  - This normally results in smoother convergence
  - This normally is faster, thanks to vectorization libraries

Gradient descent for neural nets

- For the case of the full gradients computed over all training samples, there are many ways to speed up learning esp. for smooth non-linear functions (e.g. non-linear conjugate gradients)
- Most of these techniques need adaptation for application to neural nets
- Batch gradient descent generally results in the best compromise of accuracy Vs efficiency and simplicity

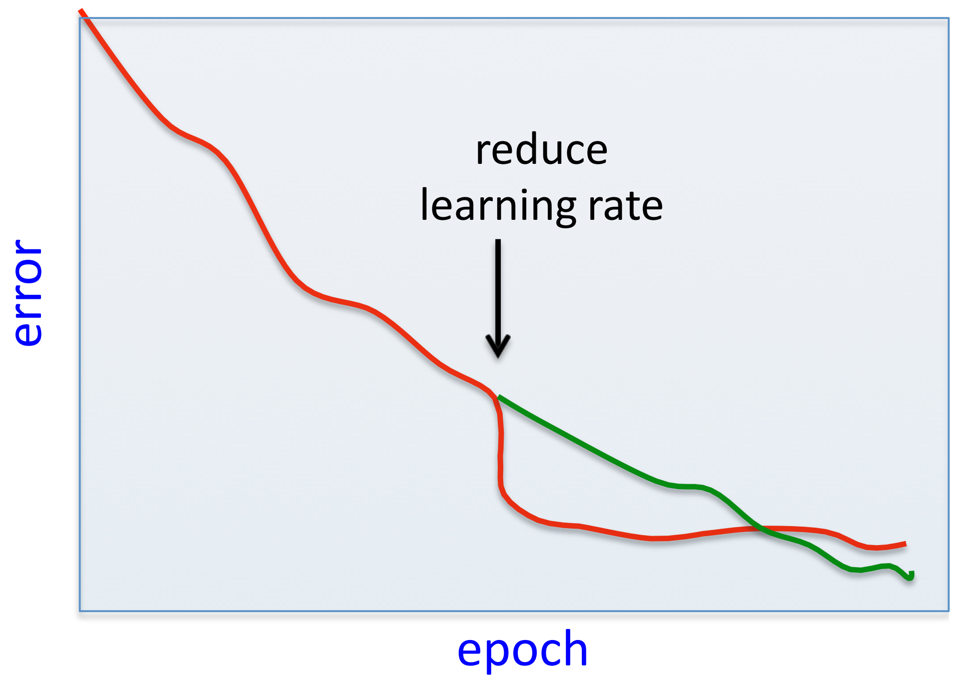
Choice of learning rate

- Guess an initial learning rate
  - If the error gets worse or oscillates wildly, reduce the learning rate
  - If the error is falling fairly consistently but slowly, increase the learning rate



Choice of learning rate, contd

- Towards the end of mini-batch learning, it nearly always helps to turn down the learning rate
  - This removes fluctuations in the final weights caused by the variations between mini-batches
- Turn down the learning rate when the error stops decreasing
  - Use the error on a separate validation set



Challenges

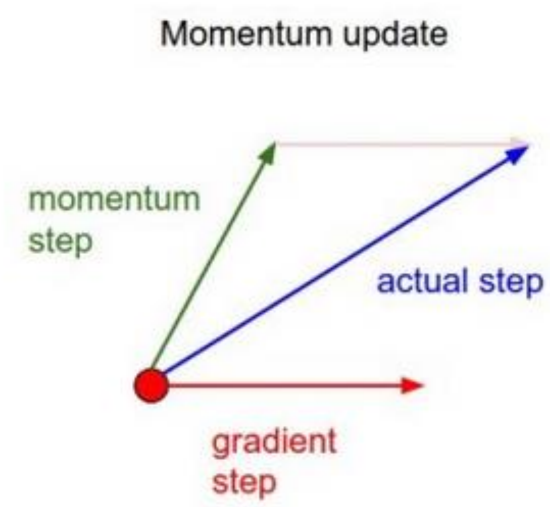
- Data challenges
  - Local Vs global minimum
  - Saddle points
- Gradient challenges
  - Vanishing or exploding gradients and convergence
- Implementation challenges
  - Needed memory
  - Precision, e.g. single- or double-precision floating point

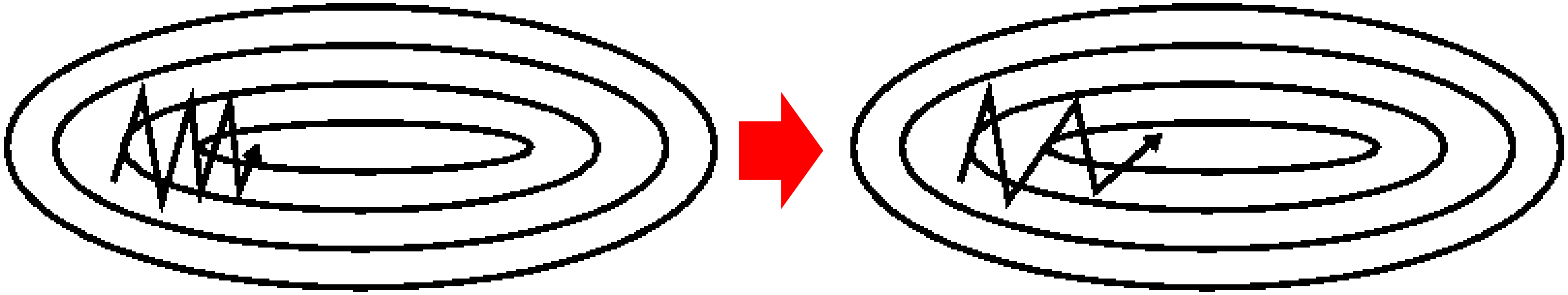
Variants of Gradient Descent

- Vanilla Gradient Descent
  - $\boldsymbol{v}^{(i)} = \nabla J(\boldsymbol{\theta}^{(i)})$
  - $\boldsymbol{\theta}^{(i+1)} = \boldsymbol{\theta}^{(i)} - \alpha\, \boldsymbol{v}^{(i)}$

Variants of Gradient Descent

- Gradient Descent with Momentum
  - $\boldsymbol{v}^{(i)} = \beta \boldsymbol{v}^{(i-1)} + (1-\beta)\, \nabla J(\boldsymbol{\theta}^{(i)})$
  - $\boldsymbol{\theta}^{(i+1)} = \boldsymbol{\theta}^{(i)} - \alpha\, \boldsymbol{v}^{(i)}$
  - Most commonly $\beta = 0.9$
- Intuition:
  - It damps oscillations in directions of high curvature by combining gradients with opposite signs
  - It builds up speed in directions with a gentle but consistent gradient





Thank you

Acknowledges: slides and material from Geoffrey Hinton, Nitish Srivastava, Kevin Swersky, Sanghyuk Chun, Marco Bressan and Fabio Golaso.