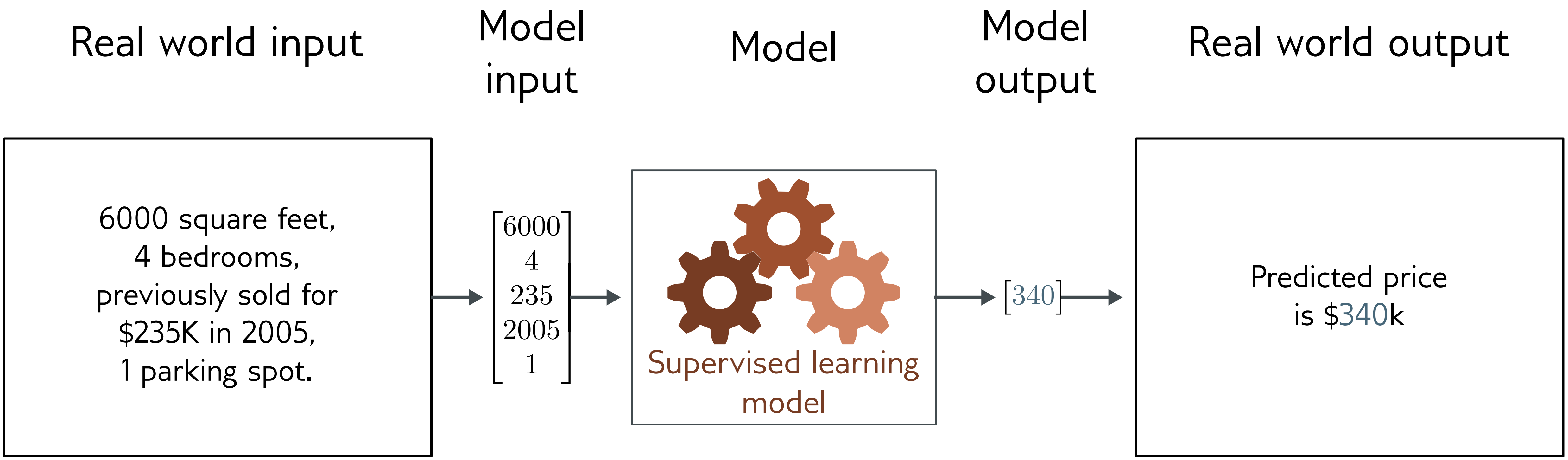
Fitting

January 22nd, 2025

Deep Learning (CS60010)

**Slides adapted from http://udlbook.com*

Regression



- Univariate regression problem (one output, real value

Loss function

- Training dataset of I pairs of input/output examples:

$$\boxed{\{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^{I}}$$

- Loss function or cost function measures how bad model is:

$$L\left[\phi, \mathrm{f}\left[\mathbf{x}_i, \phi\right], \{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^{I}\right]$$

or for short:

$$\boxed{L\left[\phi\right]}$$

Returns a scalar that is smaller when model maps inputs to outputs better

Training

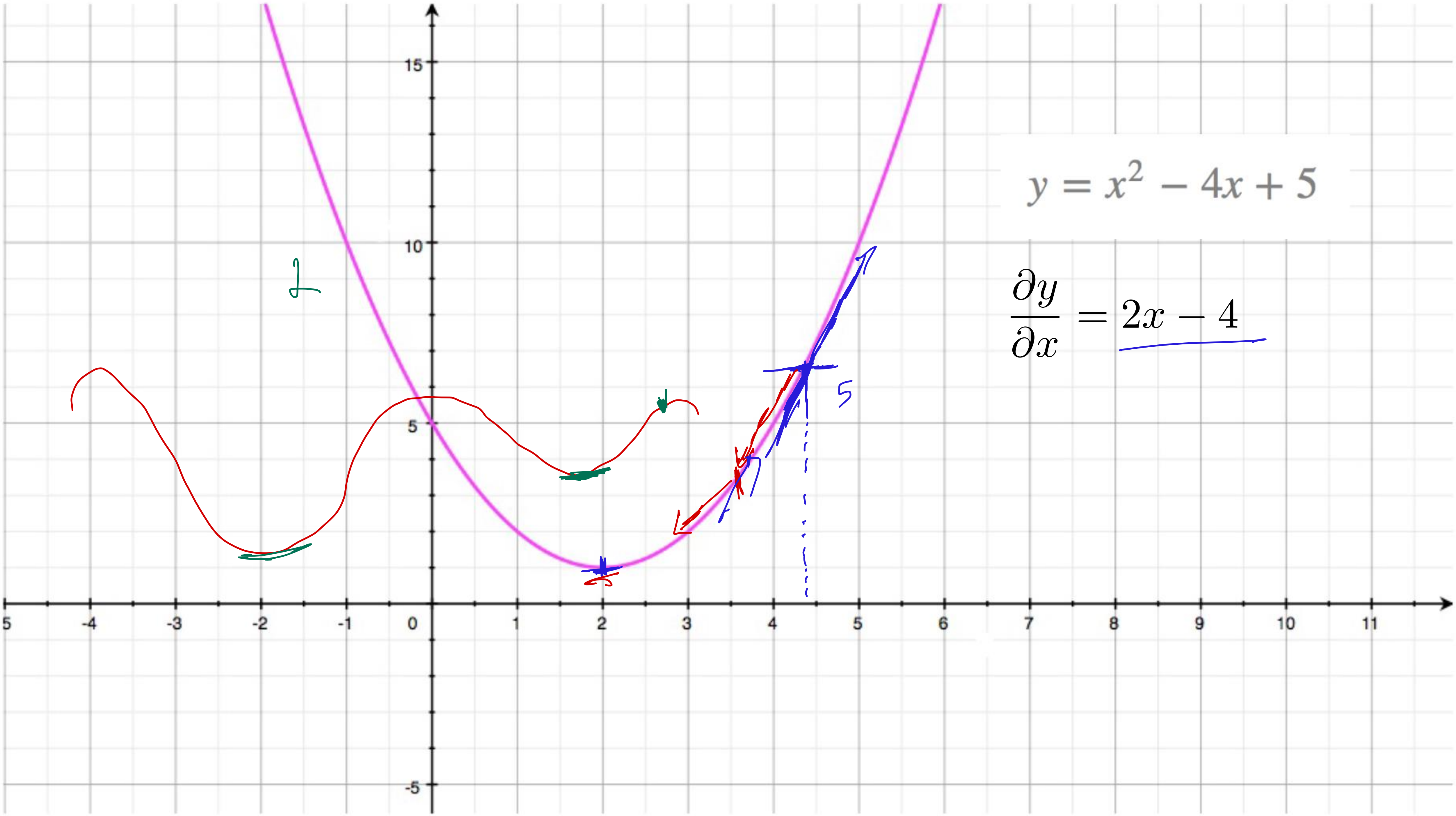
- Loss function:

$$L[\phi]$$

Returns a scalar that is smaller when model maps inputs to outputs better

- Find the parameters that minimize the loss:

$$\boxed{\hat{\phi} = \underset{\phi}{\operatorname{argmin}} \left[L[\phi] \right]}$$



$$y = x^2 - 4x + 5$$

$$\frac{\partial y}{\partial x} = 2x - 4$$

Gradient descent algorithm

Step 1. Compute the derivatives of the loss with respect to the parameters:

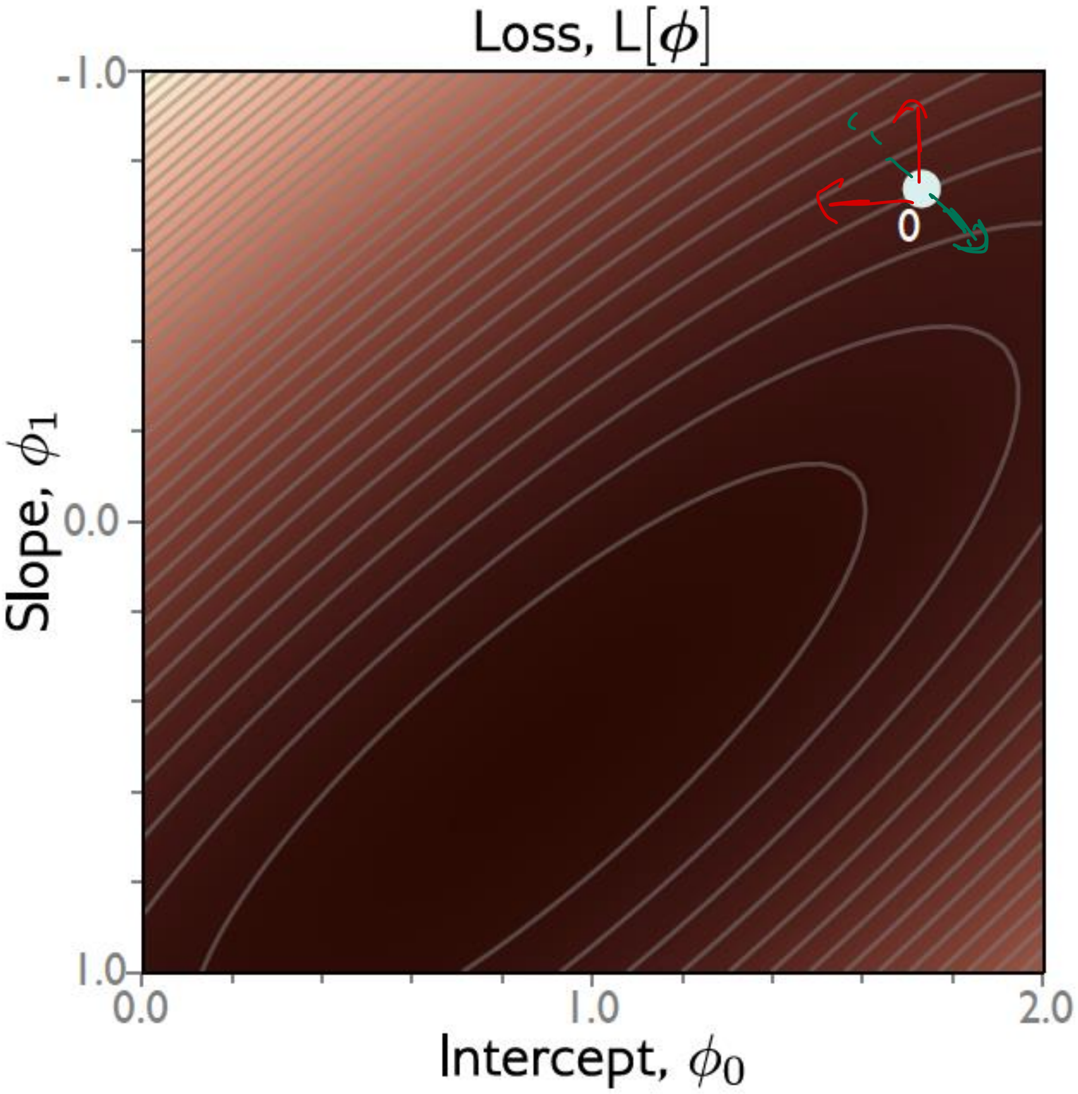
$$\frac{\partial L}{\partial \phi} = \begin{bmatrix} \frac{\partial L}{\partial \phi_0} \\ \frac{\partial L}{\partial \phi_1} \\ \vdots \\ \frac{\partial L}{\partial \phi_N} \end{bmatrix}.$$

Step 2. Update the parameters according to the rule:

$$\phi \longleftarrow \phi - \alpha \frac{\partial L}{\partial \phi},$$

where the positive scalar α determines the magnitude of the change.

Gradient descent



Step 1: Compute derivatives (slopes of function) with Respect to the parameters

$$L[\phi] = \sum_{i=1}^{I} \ell_i = \sum_{i=1}^{I} (\mathrm{f}[x_i, \phi] - y_i)^2$$
$$= \sum_{i=1}^{I} (\phi_0 + \phi_1 x_i - y_i)^2$$

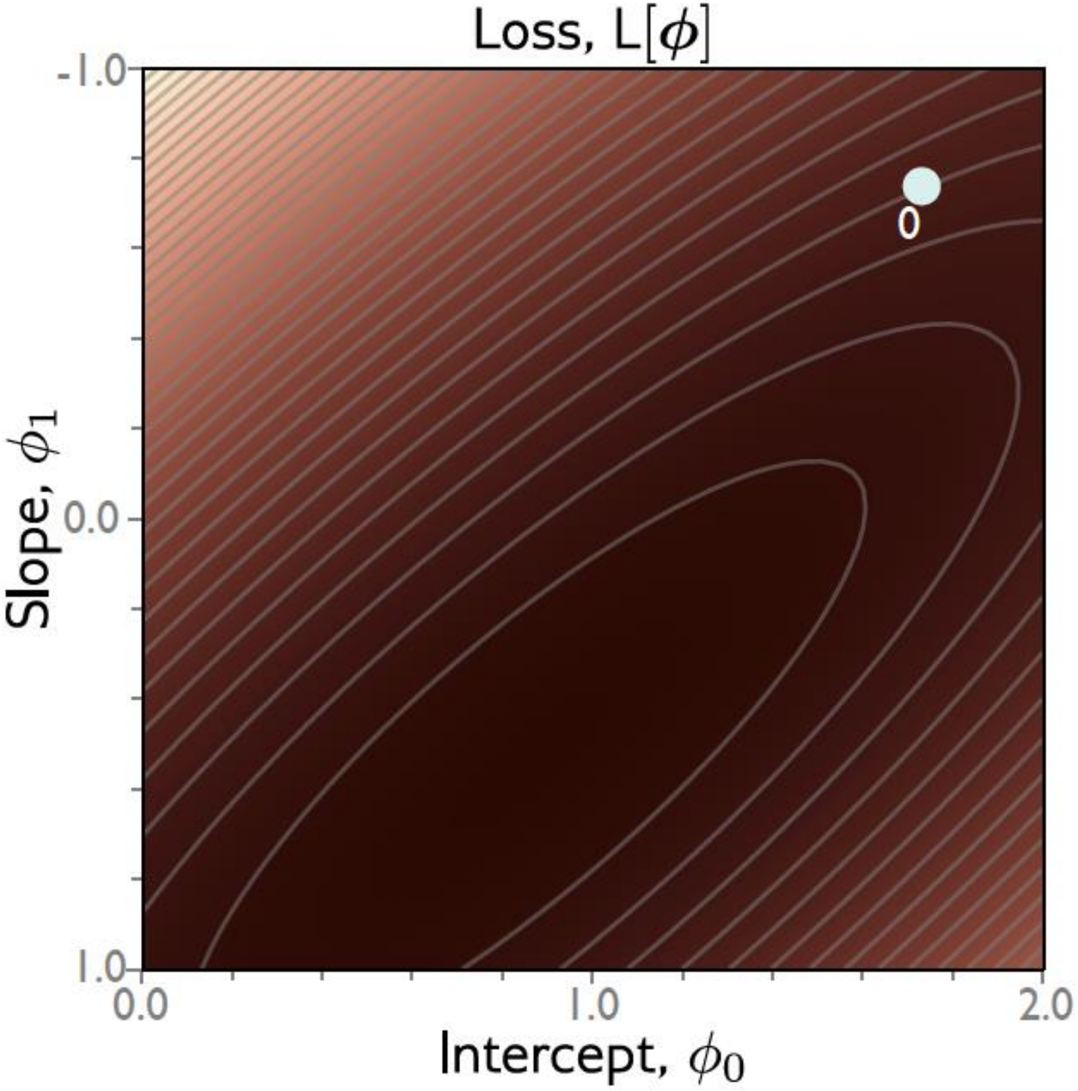
$$\frac{\partial L}{\partial \phi} = \frac{\partial}{\partial \phi} \sum_{i=1}^{I} \ell_i = \sum_{i=1}^{I} \frac{\partial \ell_i}{\partial \phi}$$

$$\frac{\partial L}{\partial \phi_0} = 2(\phi_0 + \phi_1 x_i - y_i)$$

$$\frac{\partial L}{\partial \phi_1} = 2(\phi_0 + \phi_1 x_i - y_i)\, x_i$$

=

Gradient descent



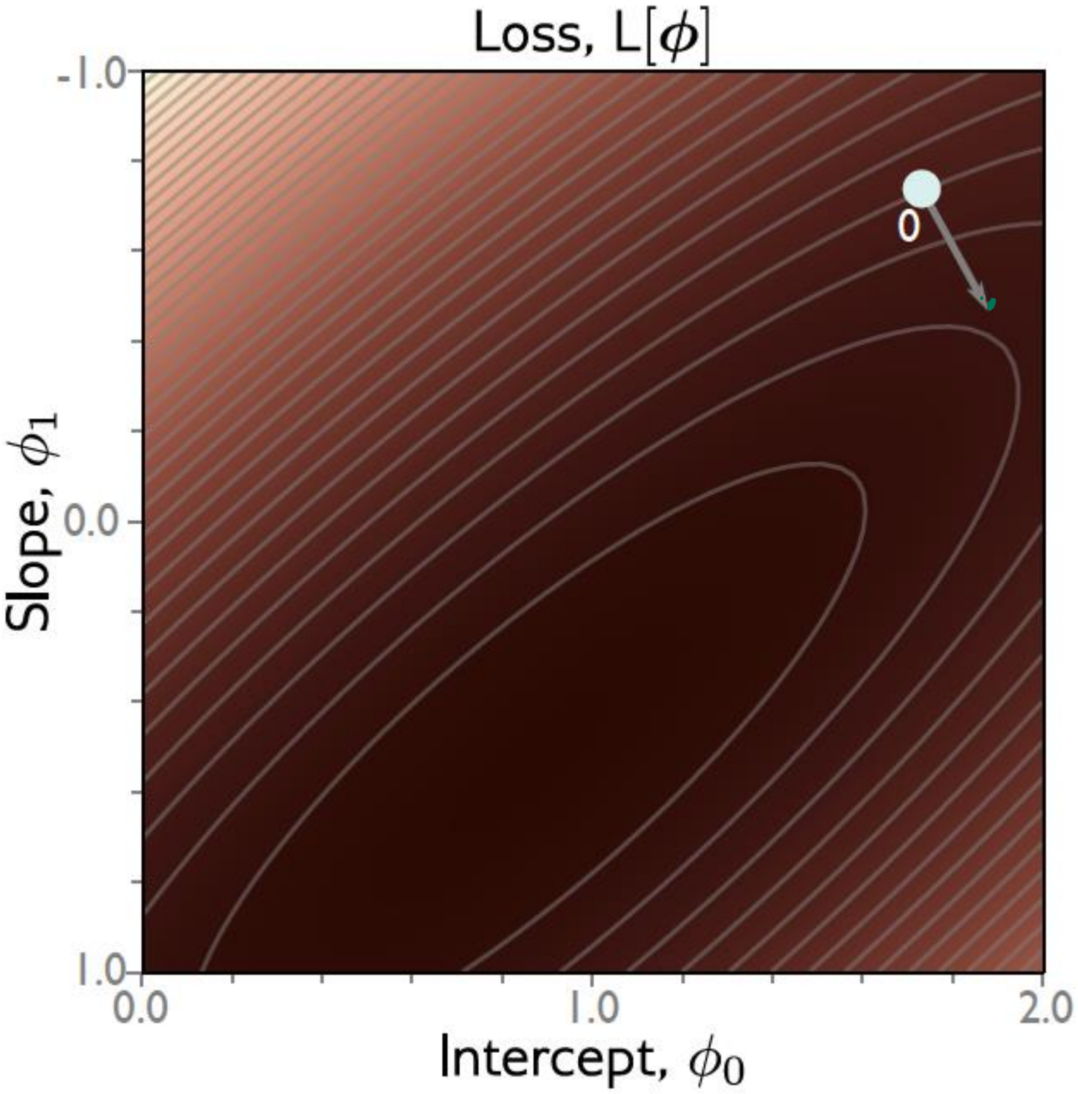
Step 1: Compute derivatives (slopes of function) with
Respect to the parameters

$$
\begin{aligned}
L[\boldsymbol{\phi}] &= \sum_{i=1}^{I} \ell_i = \sum_{i=1}^{I} \left(\mathrm{f}[x_i, \boldsymbol{\phi}] - y_i\right)^2 \\
&= \sum_{i=1}^{I} \left(\phi_0 + \phi_1 x_i - y_i\right)^2
\end{aligned}
$$

$$
\frac{\partial L}{\partial \boldsymbol{\phi}} = \frac{\partial}{\partial \boldsymbol{\phi}} \sum_{i=1}^{I} \ell_i = \sum_{i=1}^{I} \frac{\partial \ell_i}{\partial \boldsymbol{\phi}}
$$

$$
\frac{\partial \ell_i}{\partial \boldsymbol{\phi}} = \begin{bmatrix} \frac{\partial \ell_i}{\partial \phi_0} \\ \frac{\partial \ell_i}{\partial \phi_1} \end{bmatrix} = \begin{bmatrix} 2(\phi_0 + \phi_1 x_i - y_i) \\ 2x_i(\phi_0 + \phi_1 x_i - y_i) \end{bmatrix}
$$

Gradient descent



Step 1: Compute derivatives (slopes of function) with Respect to the parameters

$$\frac{\partial L}{\partial \phi} = \frac{\partial}{\partial \phi} \sum_{i=1}^{I} \ell_i = \sum_{i=1}^{I} \frac{\partial \ell_i}{\partial \phi}$$

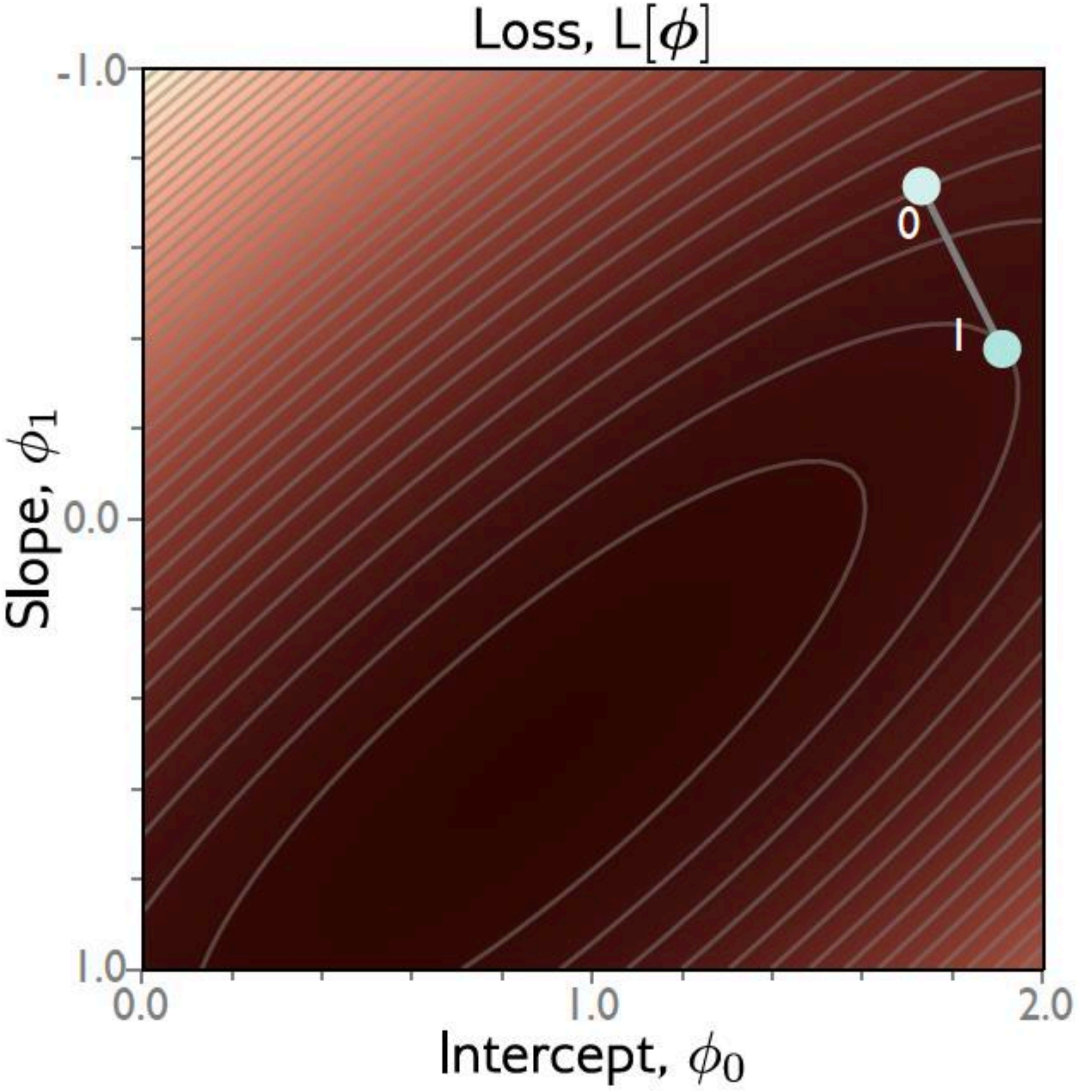
$$\frac{\partial \ell_i}{\partial \phi} = \begin{bmatrix} \frac{\partial \ell_i}{\partial \phi_0} \\ \frac{\partial \ell_i}{\partial \phi_1} \end{bmatrix} = \begin{bmatrix} 2(\phi_0 + \phi_1 x_i - y_i) \\ 2x_i(\phi_0 + \phi_1 x_i - y_i) \end{bmatrix}$$

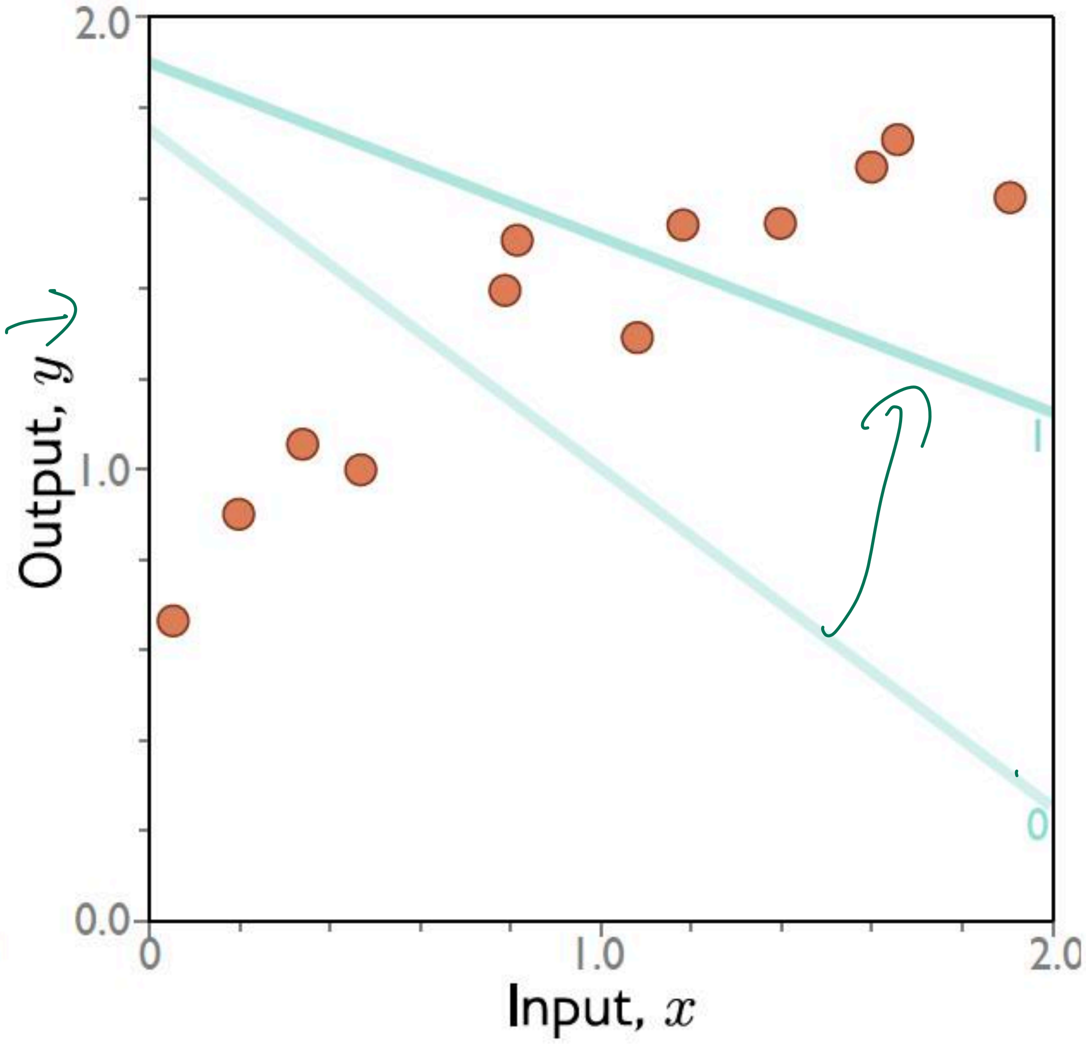
Step 2: Update parameters according to rule

$$\phi \longleftarrow \phi - \alpha \frac{\partial L}{\partial \phi}$$

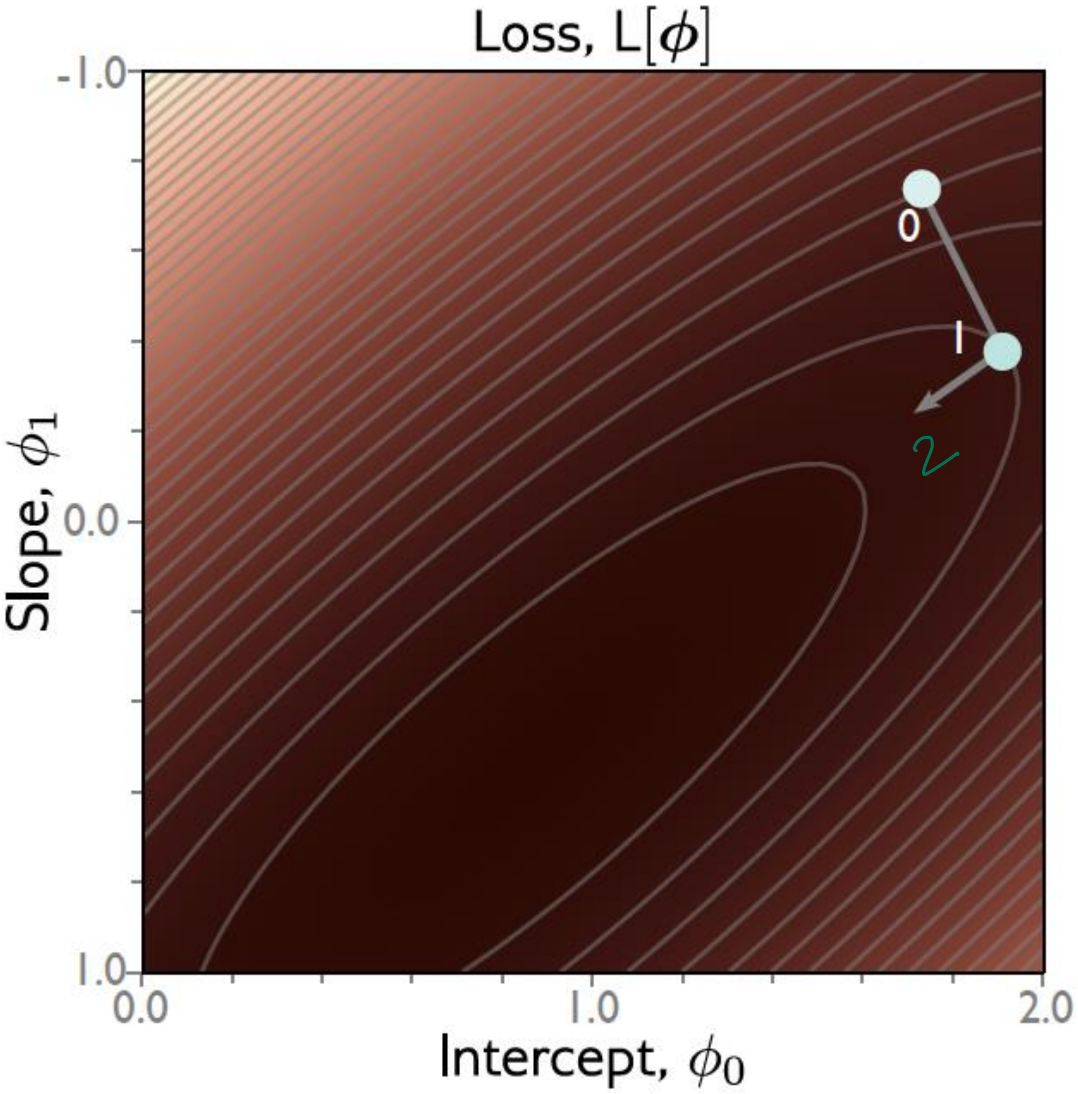
α = step size or learning rate if fixed

Gradient descent





Gradient descent



Step 1: Compute derivatives (slopes of function) with Respect to the parameters

$$\frac{\partial L}{\partial \phi} = \frac{\partial}{\partial \phi}\sum_{i=1}^{I} \ell_i = \sum_{i=1}^{I} \frac{\partial \ell_i}{\partial \phi}$$

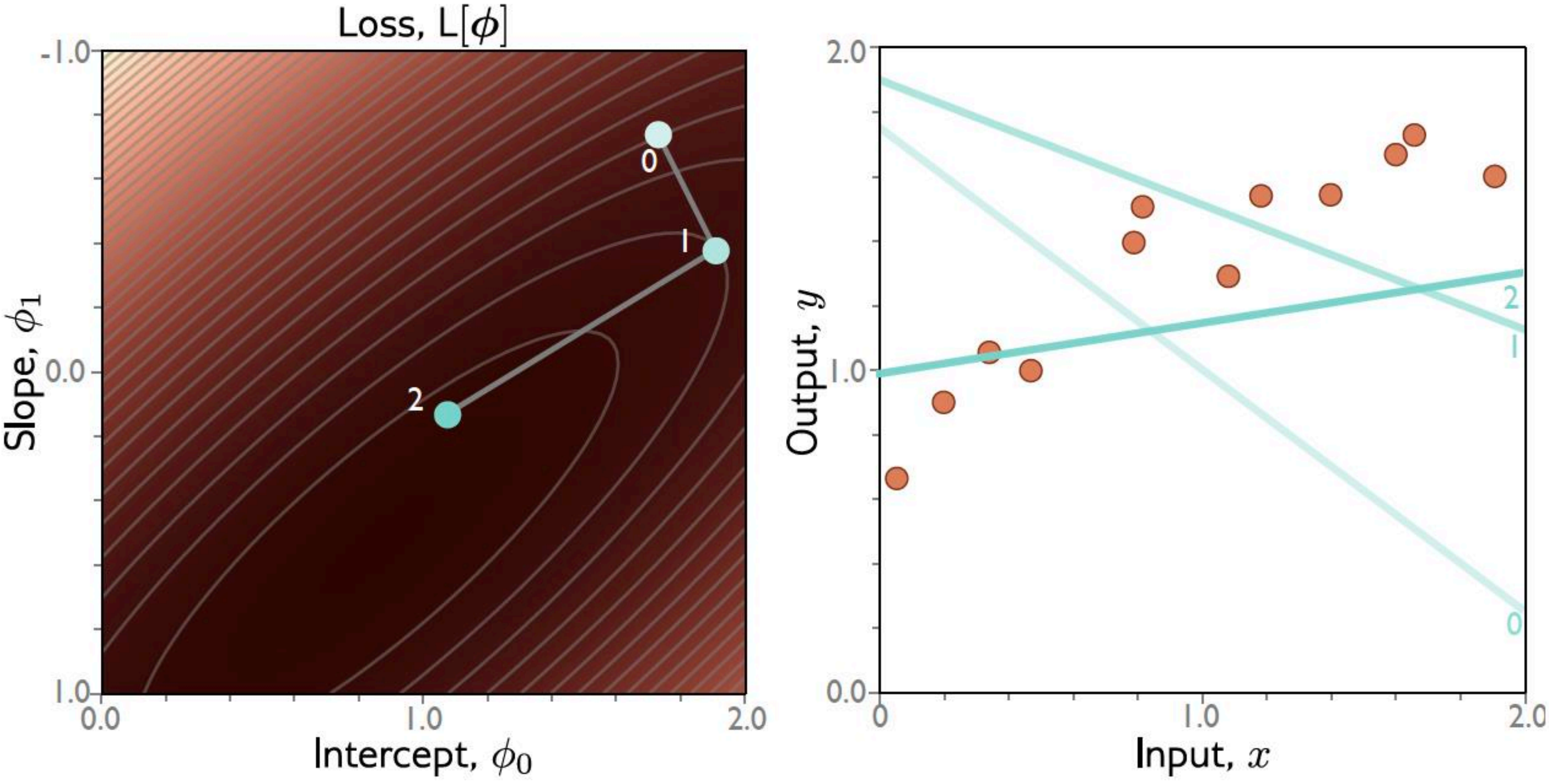
$$\frac{\partial \ell_i}{\partial \boldsymbol{\phi}} = \begin{bmatrix} \frac{\partial \ell_i}{\partial \phi_0} \\ \frac{\partial \ell_i}{\partial \phi_1} \end{bmatrix} = \begin{bmatrix} 2(\phi_0 + \phi_1 x_i - y_i) \\ 2x_i(\phi_0 + \phi_1 x_i - y_i) \end{bmatrix}$$

Step 2: Update parameters according to rule

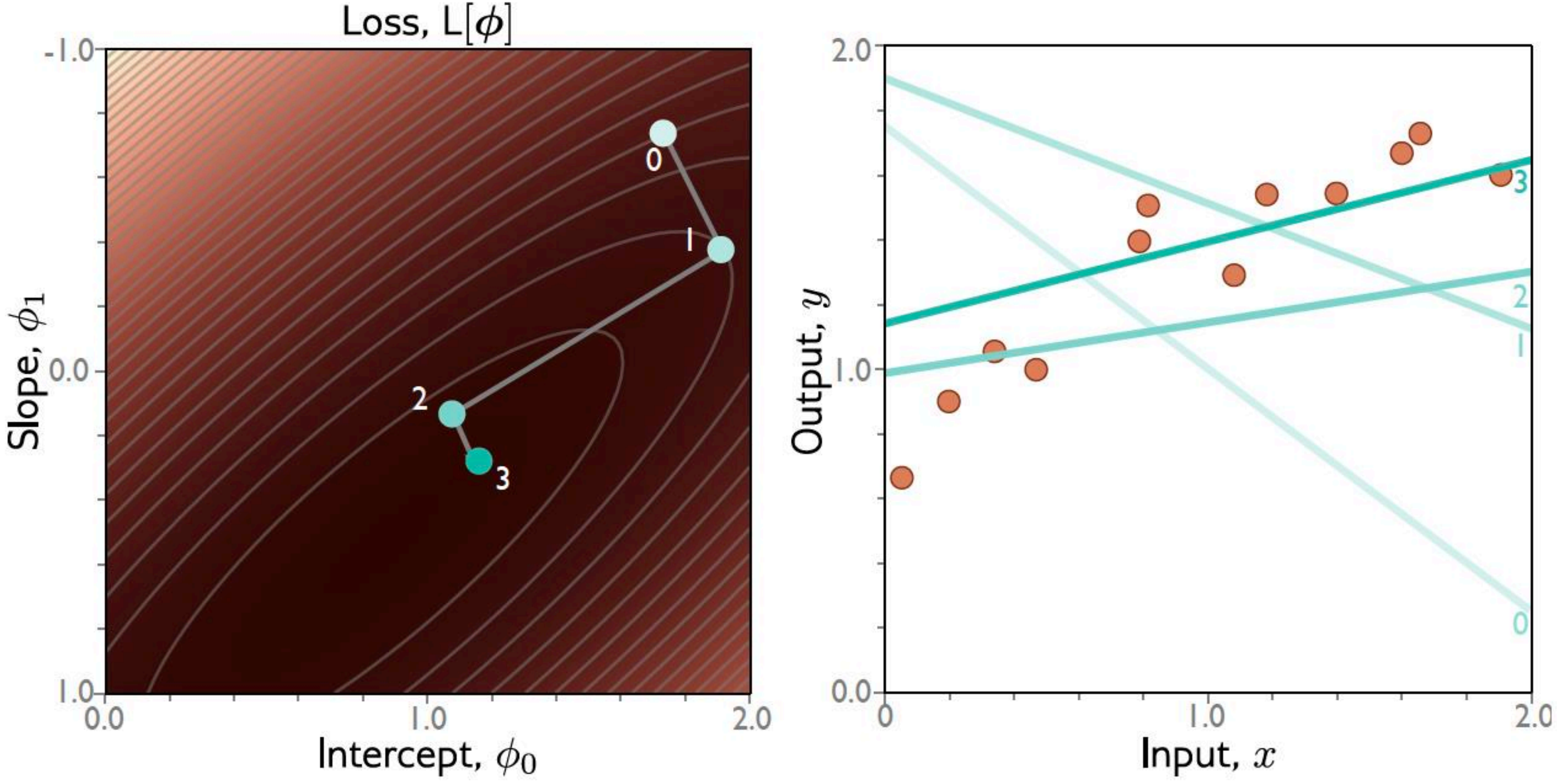
$$\boldsymbol{\phi} \longleftarrow \boldsymbol{\phi} - \alpha \frac{\partial L}{\partial \boldsymbol{\phi}}$$

α = step size

Gradient descent

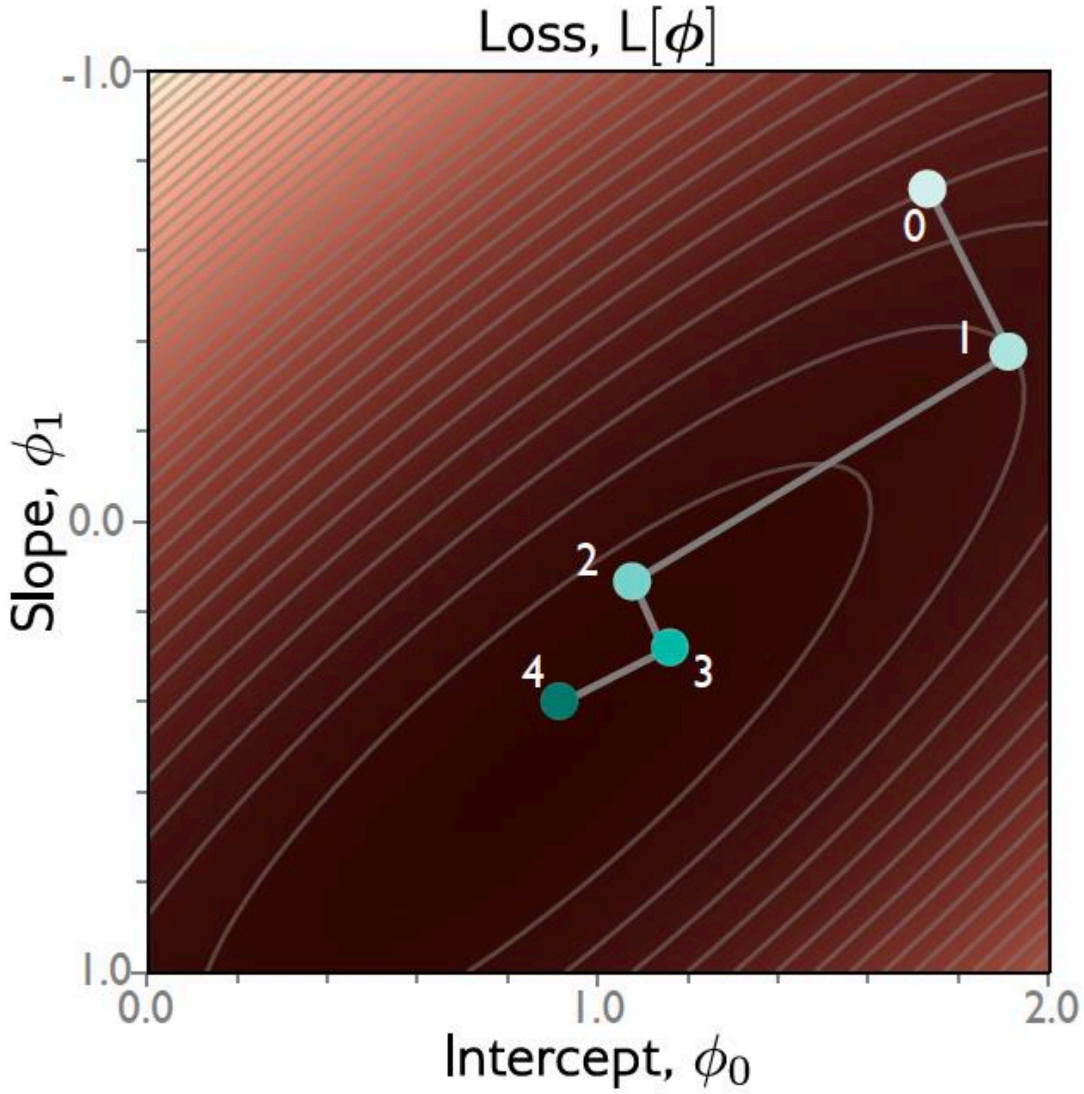


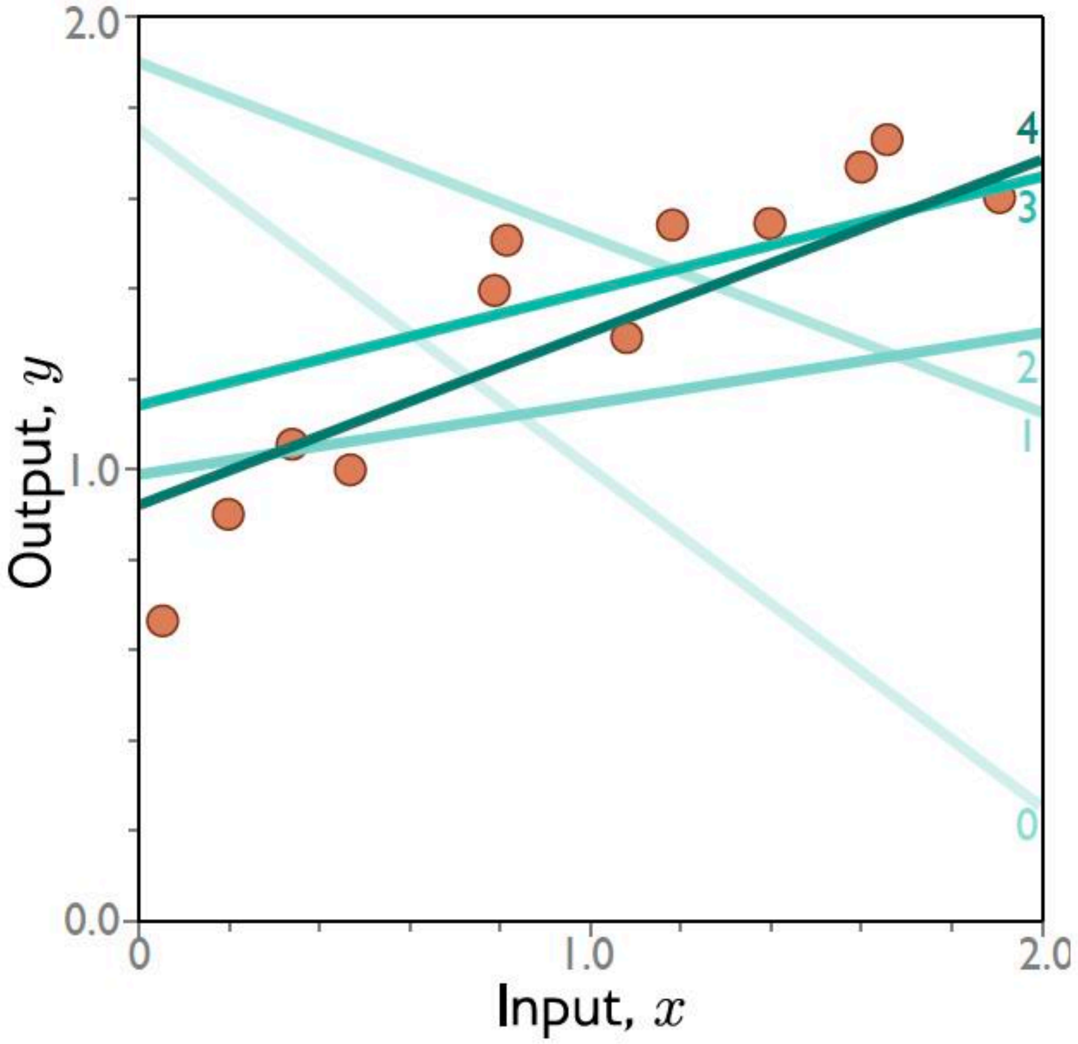
Gradient descent



Gradient descent

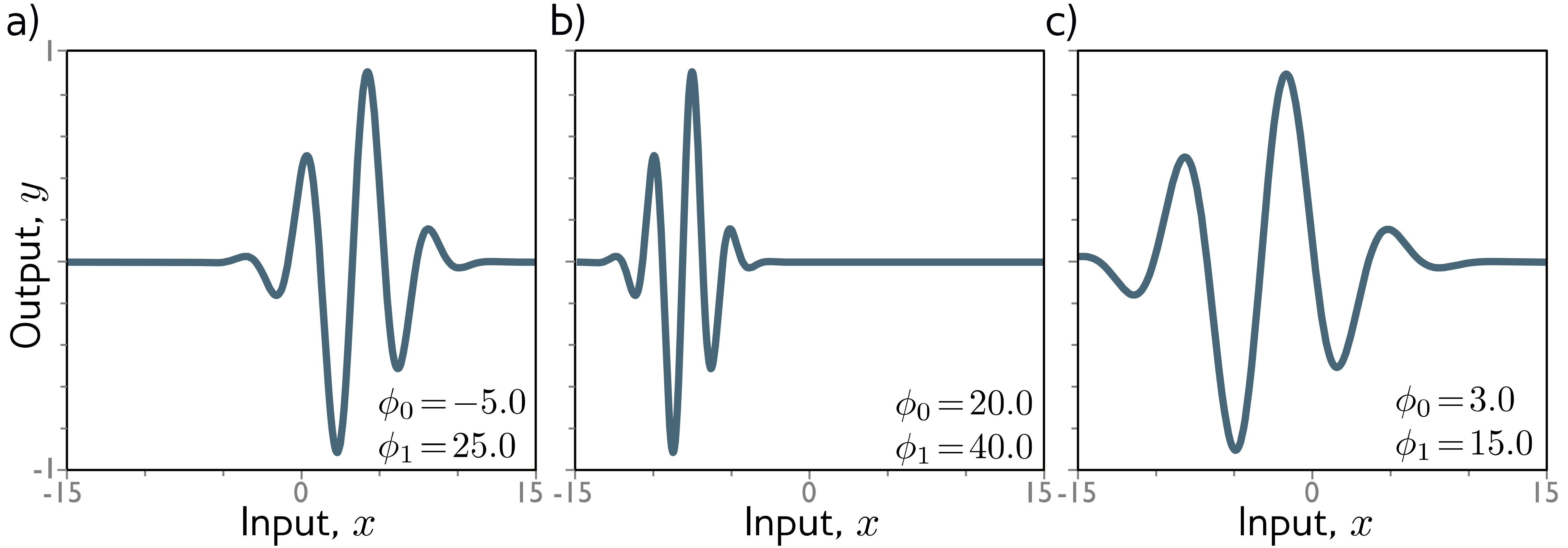
Converge





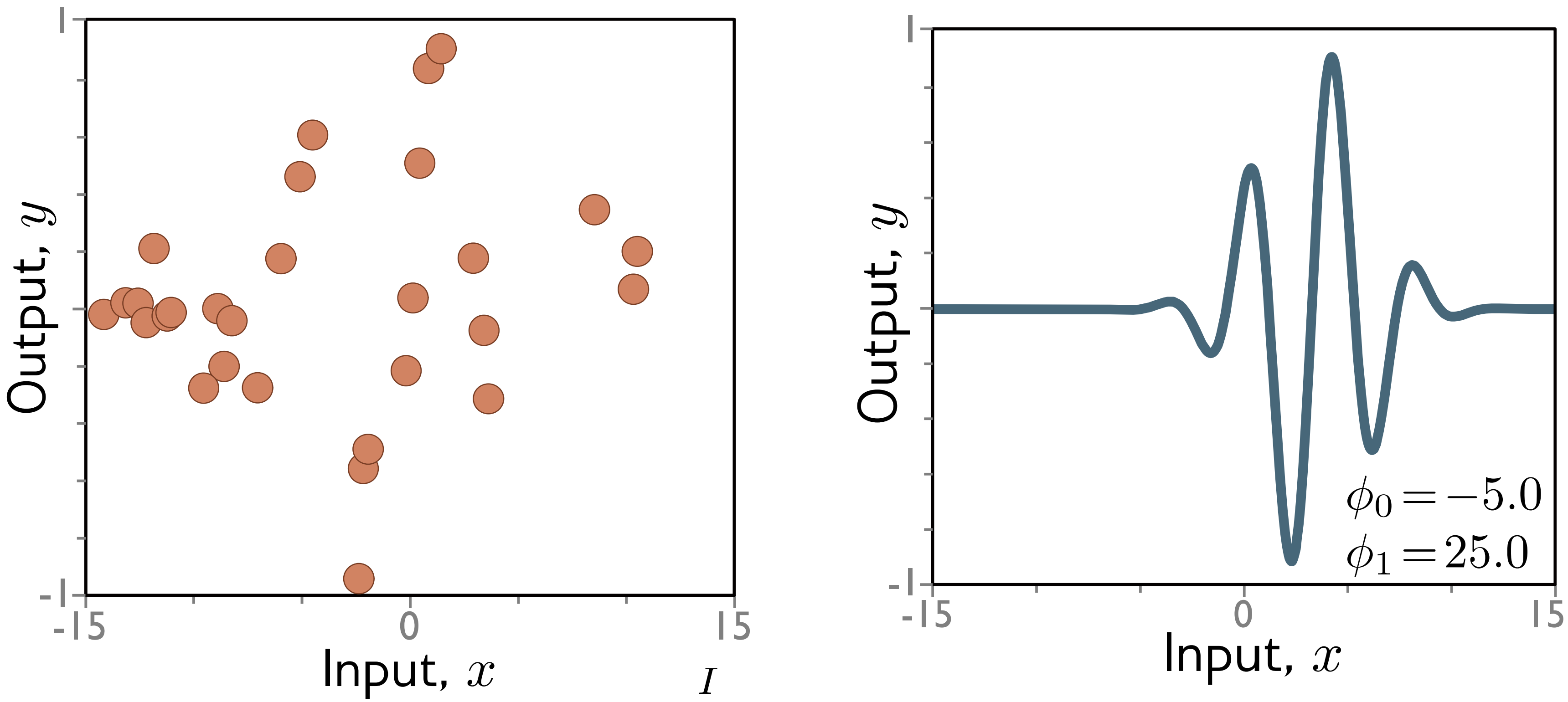
Gabor model

$$\mathrm{f}[x, \boldsymbol{\phi}] = \sin[\phi_0 + 0.06 \cdot \phi_1 x] \cdot \exp\left(-\frac{(\phi_0 + 0.06 \cdot \phi_1 x)^2}{8.0}\right)$$

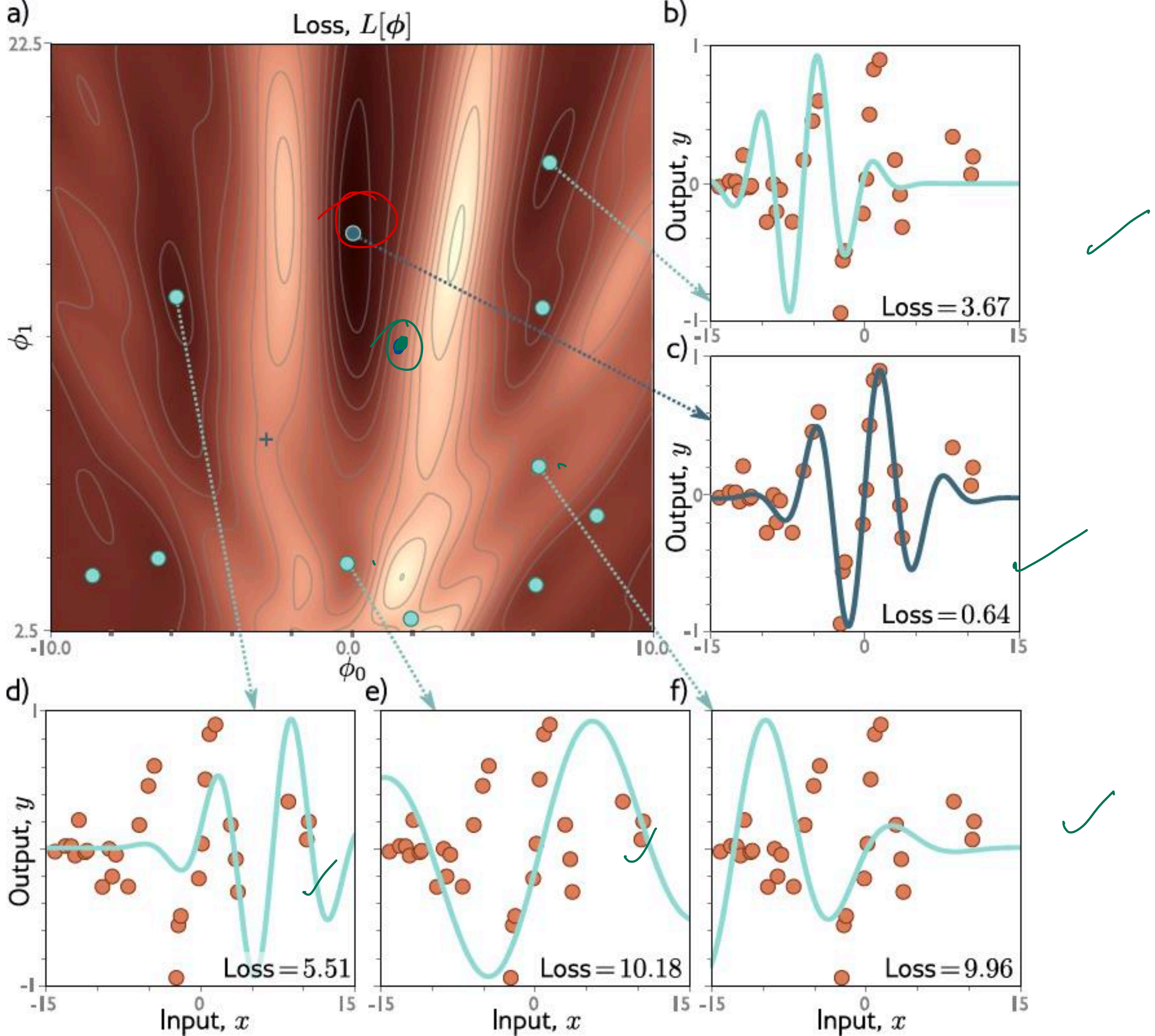


Gabor model

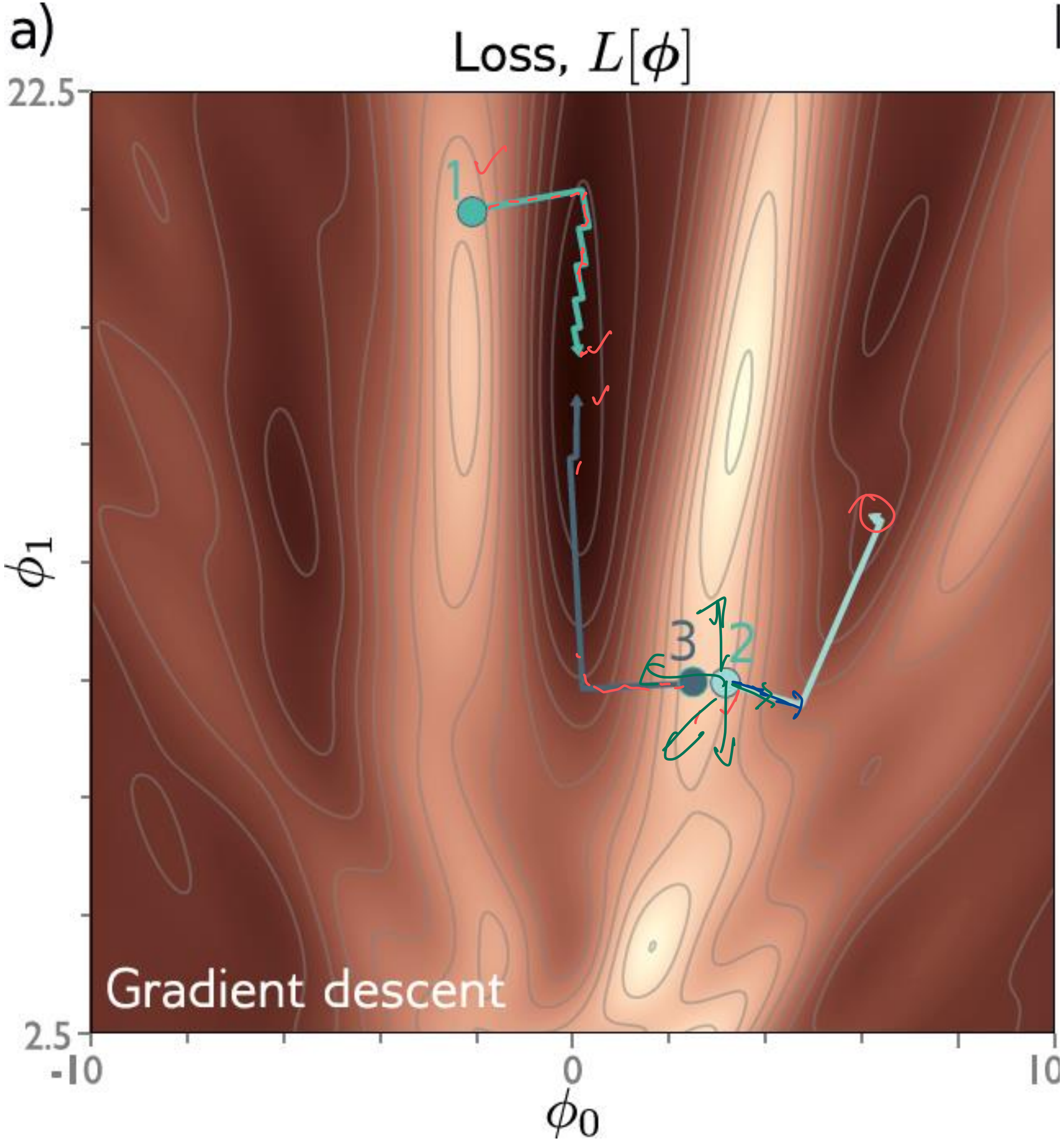
$$\mathrm{f}[x, \boldsymbol{\phi}] = \sin[\phi_0 + 0.06 \cdot \phi_1 x] \cdot \exp\left(-\frac{(\phi_0 + 0.06 \cdot \phi_1 x)^2}{8.0}\right)$$



$$L[\boldsymbol{\phi}] = \sum_{i=1}^{I} \left(\mathrm{f}[x_i, \boldsymbol{\phi}] - y_i\right)^2$$



a)
Loss, $L[\phi]$
22.5
2.5
ϕ_1
-10.0
0.0
10.0
ϕ_0
b)
Loss = 3.67
c)
Loss = 0.64
d)
Loss = 5.51
e)
Loss = 10.18
f)
Loss = 9.96
Output, y
Input, x
1
0
-1
-15
0
15



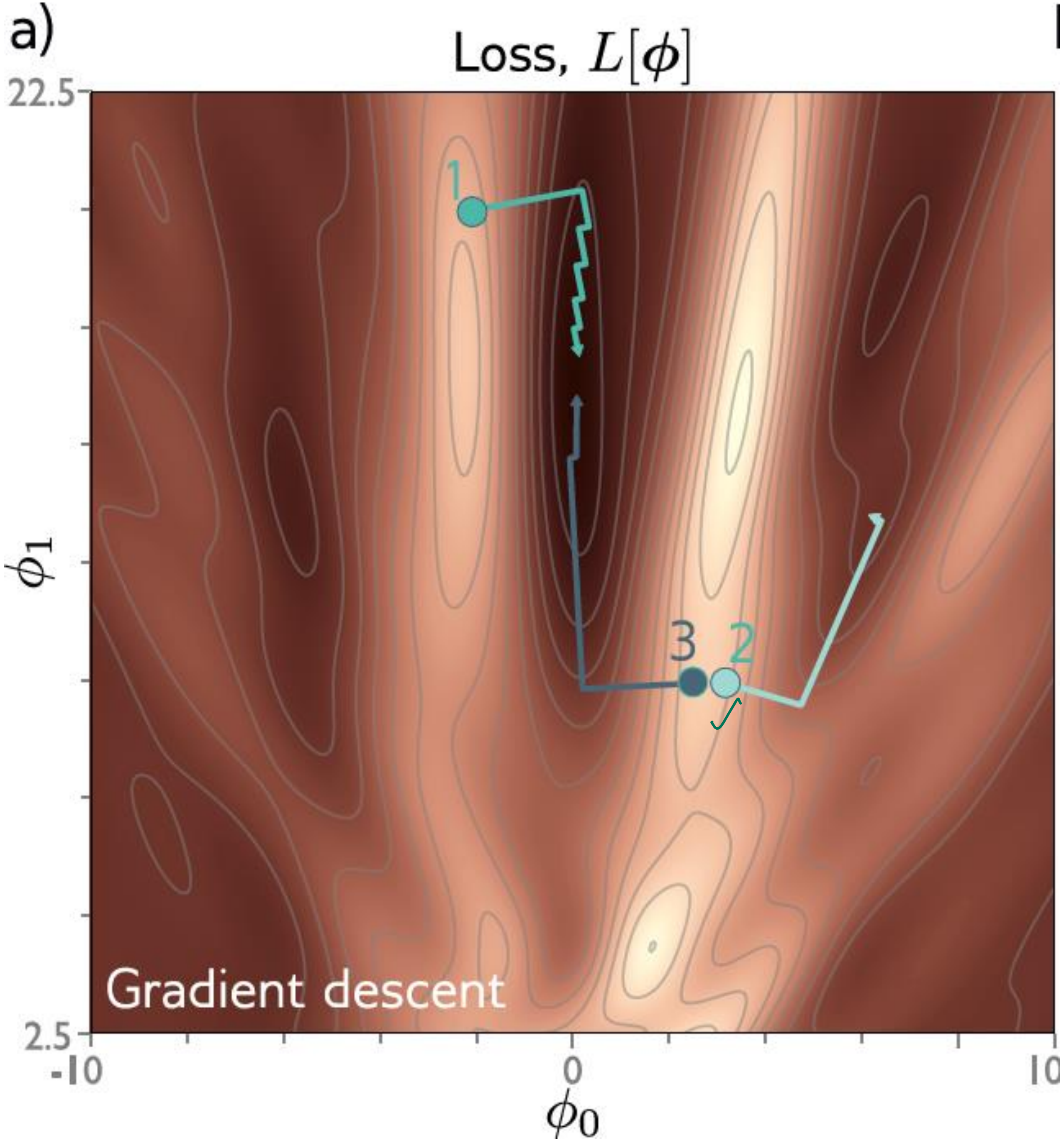
$\boxed{1000}$

25 25 — — 25

- Gradient descent gets to the global minimum if we start in the right "valley"
- Otherwise, descent to a local minimum
- Or get stuck near a saddle point

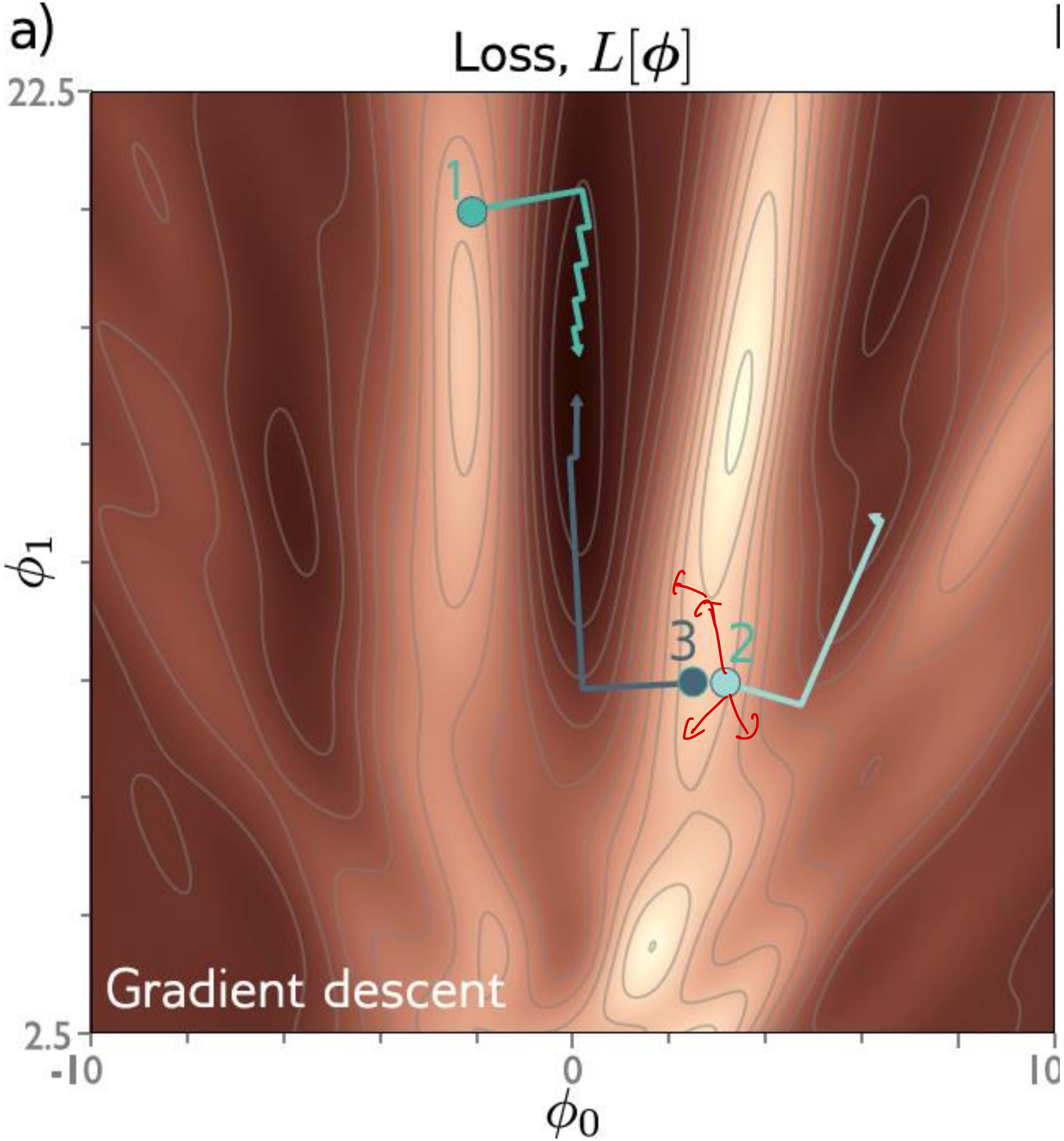
Fitting models

- Maths overview
- Gradient descent algorithm
- Linear regression example
- Gabor model example
- Stochastic gradient descent
- Momentum
- Adam



IDEA: add noise

- Stochastic gradient descent
- Compute gradient based on only a subset of points – a mini-batch
- Work through dataset sampling without replacement
- One pass though the data is called an epoch



Stochastic gradient descent

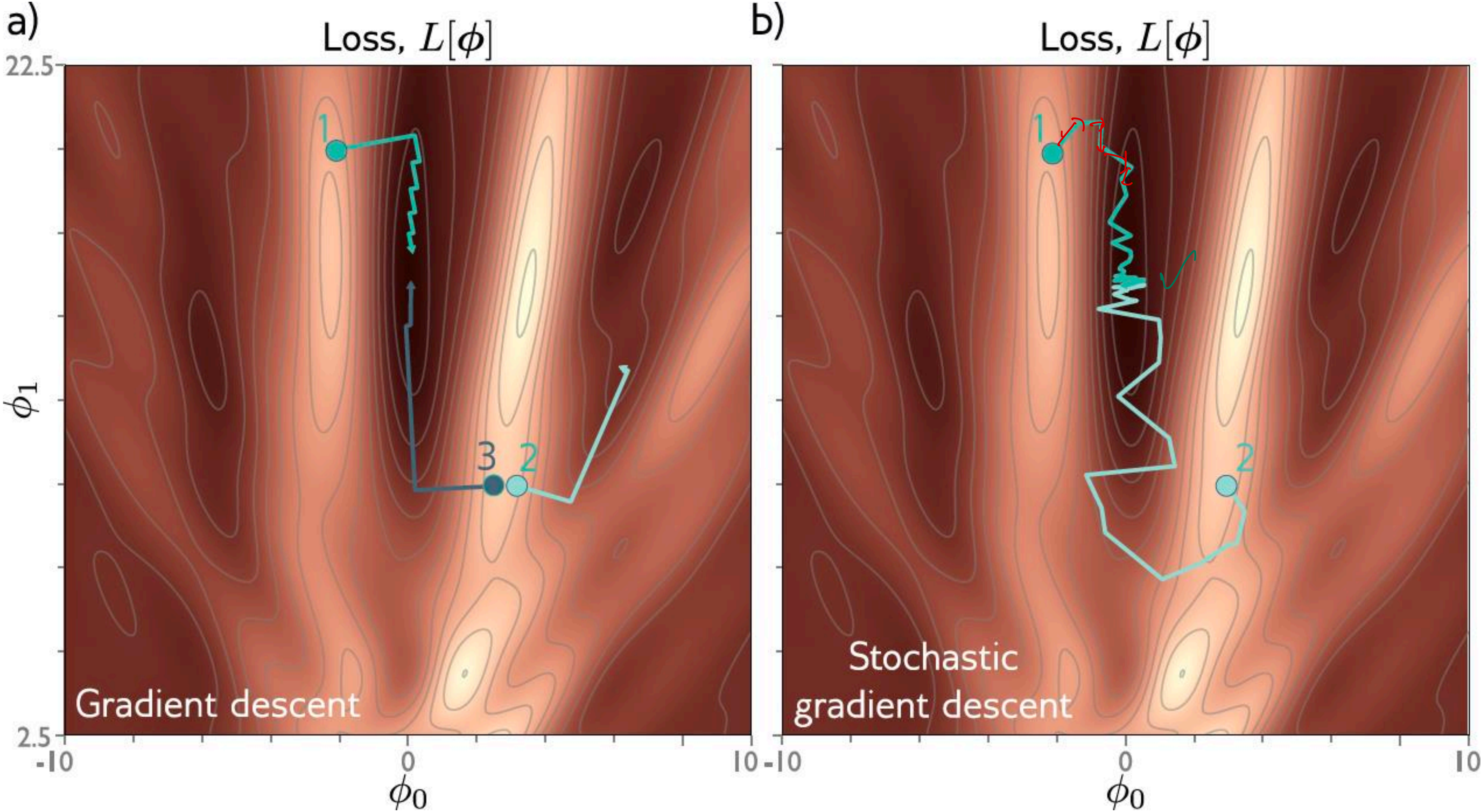
Before (full batch descent)

$$\boxed{\phi_{t+1} \longleftarrow \phi_t - \alpha \sum_{i=1}^{I} \frac{\partial \ell_i[\phi_t]}{\partial \phi},}$$

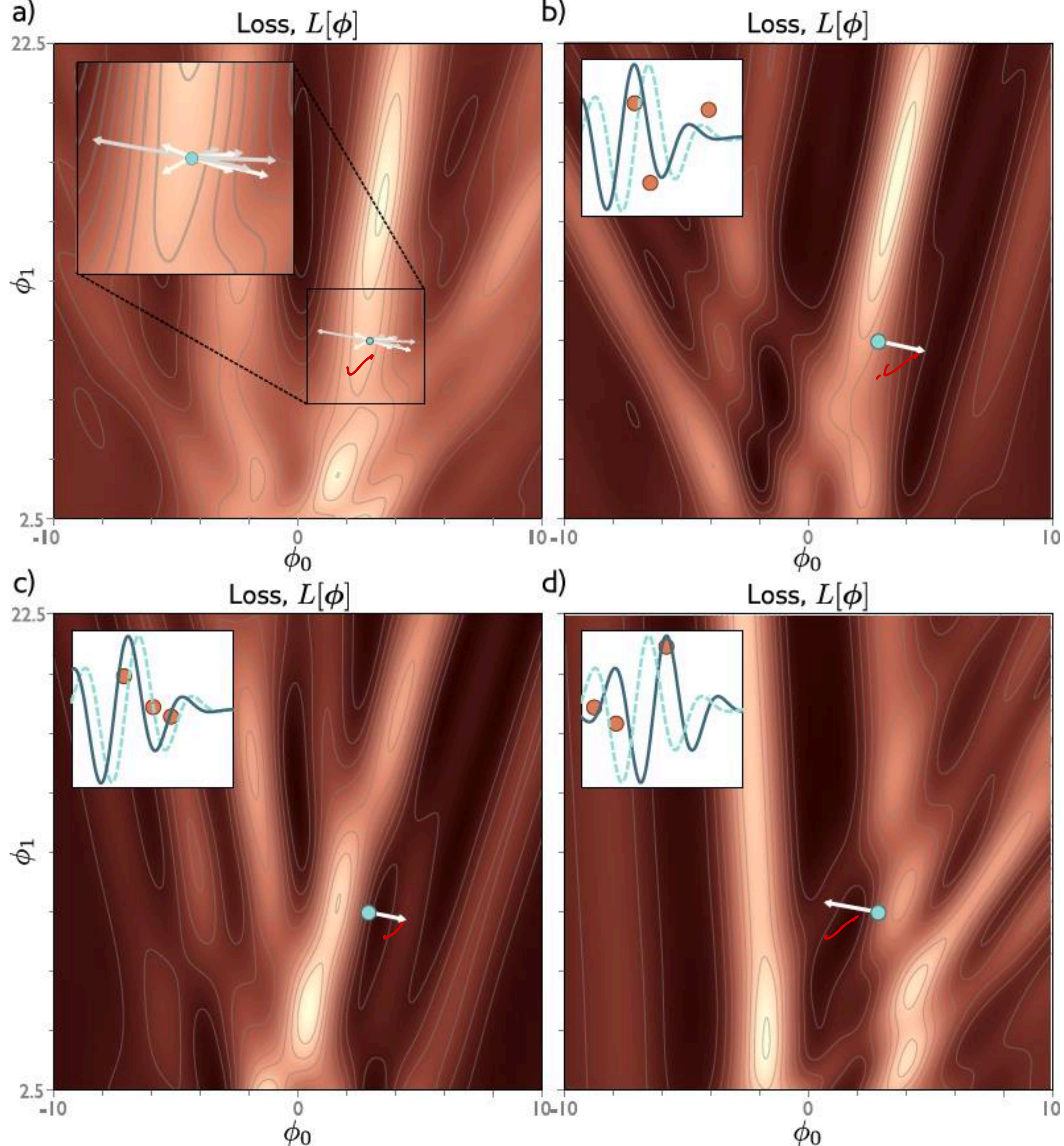
After (SGD)

$$\phi_{t+1} \longleftarrow \phi_t - \alpha \sum_{i \in \mathcal{B}_t} \frac{\partial \ell_i[\phi_t]}{\partial \phi},$$

Fixed learning rate α



a)
Loss, $L[\phi]$
22.5
2.5
-10
0
10
ϕ_1
ϕ_0
1
2
3
Gradient descent
b)
Loss, $L[\phi]$
-10
0
10
ϕ_0
1
2
Stochastic gradient descent


a)
Loss, $L[\phi]$
b)
Loss, $L[\phi]$
c)
Loss, $L[\phi]$
d)
Loss, $L[\phi]$
22.5
2.5
ϕ_1
ϕ_0
-10
0
10

Properties of SGD

- Can escape from local minima
- Adds noise, but still sensible updates as based on part of data
- Uses all data equally
- Less computationally expensive
- Seems to find better solutions

- Doesn't converge in traditional sense
- Learning rate schedule – decrease learning rate over time

Fitting models

- Maths overview
- Gradient descent algorithm
- Linear regression example
- Gabor model example
- Stochastic gradient descent
- Momentum
- Adam

Momentum: Towards a smoother trajectory

- Weighted sum of this gradient and previous gradient

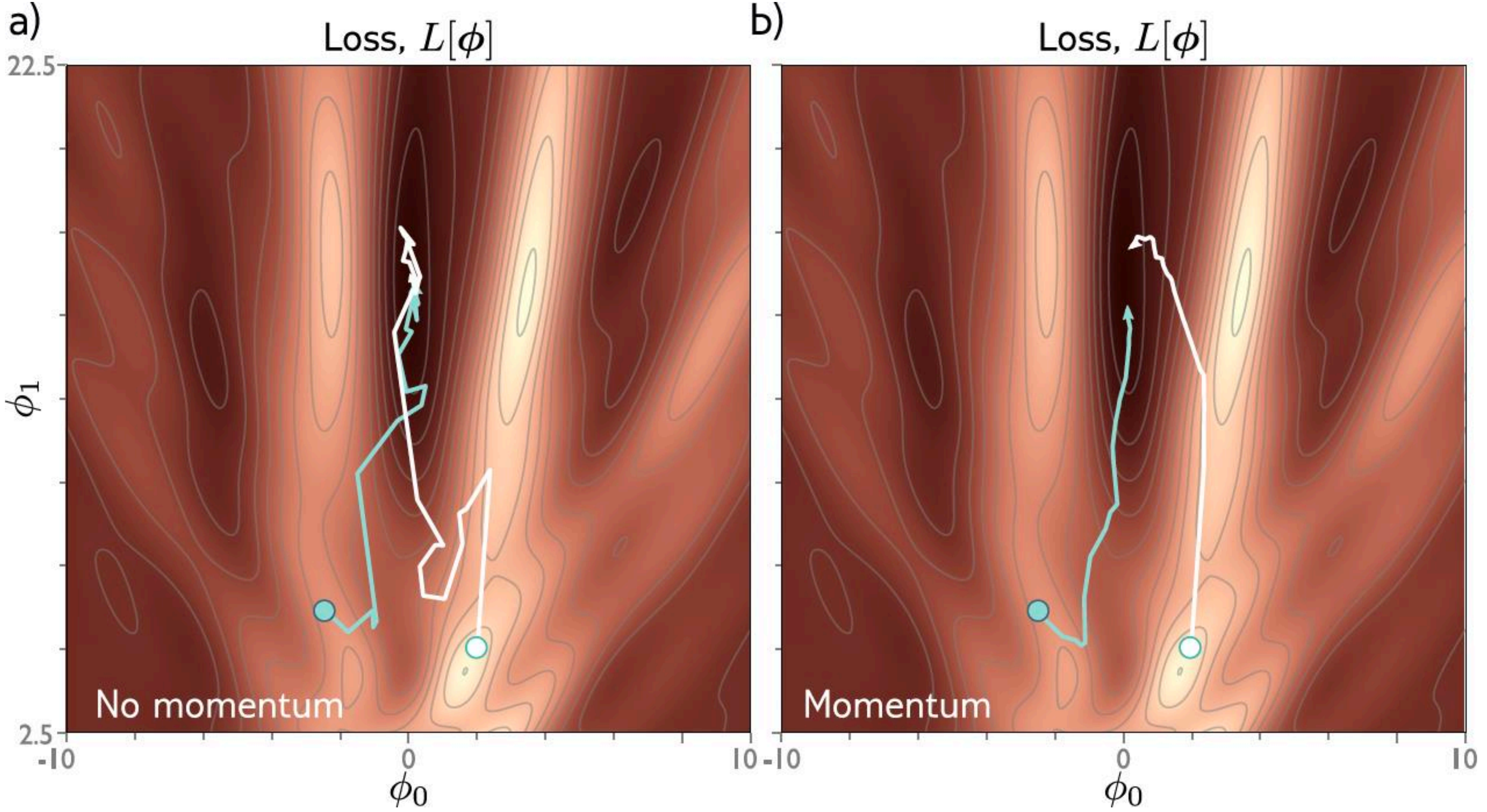
$$\mathbf{m}_{t+1} \leftarrow \beta \cdot \mathbf{m}_t + (1-\beta) \boxed{\sum_{i \in \mathcal{B}_t} \frac{\partial \ell_i[\boldsymbol{\phi}_t]}{\partial \boldsymbol{\phi}}}$$

$$\boxed{\boldsymbol{\phi}_{t+1} \leftarrow \boldsymbol{\phi}_t - \alpha \cdot \mathbf{m}_{t+1}}$$

$m_{t+1} = \beta m_t + —$

$m_t = \beta m_{t-1} + - -$

$m_{t+1} = \frac{\partial \ell_i[\phi_t]}{\partial \phi}$



1. Less dependence on learning rate. 2. Converges faster

Nesterov accelerated momentum

- Momentum is kind of like a prediction of where we are going

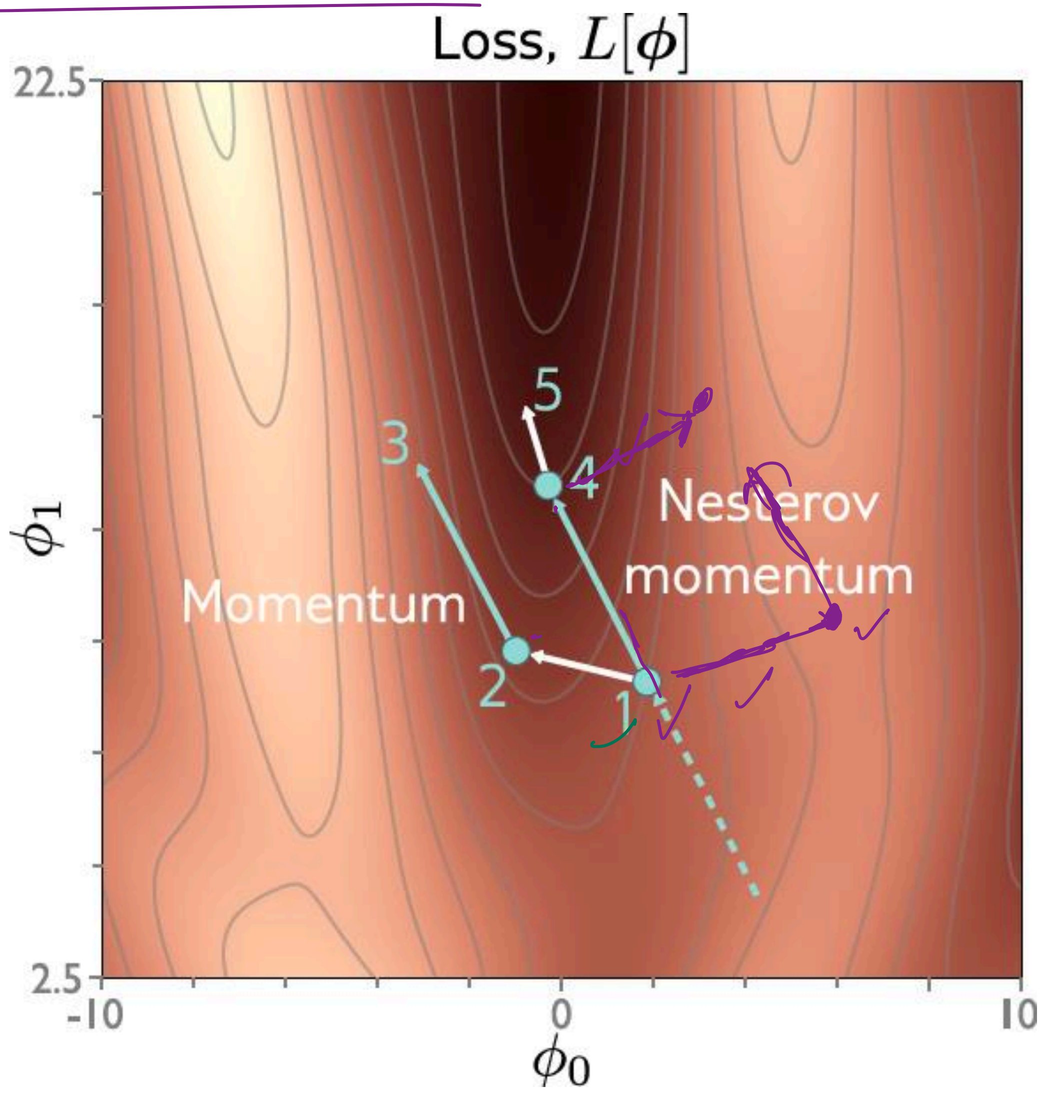
$$\mathbf{m}_{t+1} \leftarrow \beta \cdot \mathbf{m}_t + (1-\beta) \sum_{i \in \mathcal{B}_t} \frac{\partial \ell_i[\boldsymbol{\phi}_t]}{\partial \boldsymbol{\phi}}$$

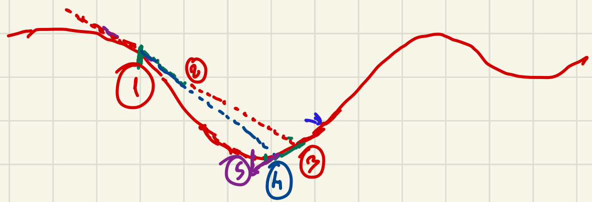
$$\boldsymbol{\phi}_{t+1} \leftarrow \boldsymbol{\phi}_t - \alpha \cdot \mathbf{m}_{t+1}$$

- Move in the predicted direction, THEN, measure the gradient

$$\mathbf{m}_{t+1} \leftarrow \beta \cdot \mathbf{m}_t + (1-\beta) \sum_{i \in \mathcal{B}_t} \frac{\partial \ell_i[\boldsymbol{\phi}_t - \alpha \cdot \mathbf{m}_t]}{\partial \boldsymbol{\phi}}$$

$$\boldsymbol{\phi}_{t+1} \leftarrow \boldsymbol{\phi}_t - \alpha \cdot \mathbf{m}_{t+1}$$





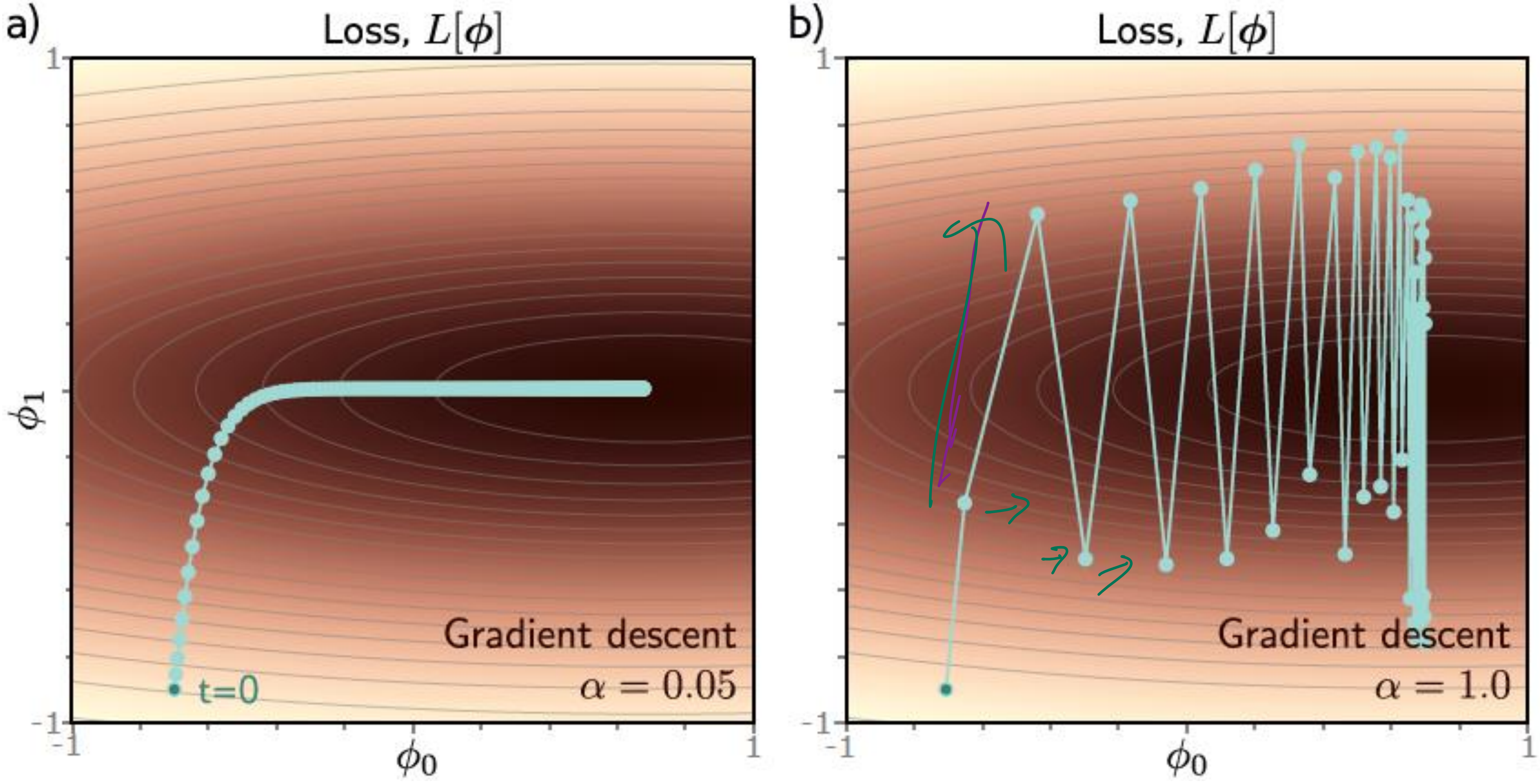
2, 3

→ Momentum

Fitting models

- Maths overview
- Gradient descent algorithm
- Linear regression example
- Gabor model example
- Stochastic gradient descent
- Momentum
- Adam

Adaptive moment estimation (Adam)



Gradient may be much steeper in one direction than another, making it difficult to choose a good learning rate that makes good progress in both the directions, and is stable

Normalized gradients

- Measure mean and pointwise squared gradient

$$\mathbf{m}_{t+1} \leftarrow \frac{\partial L[\phi_t]}{\partial \phi}$$

$$\mathbf{v}_{t+1} \leftarrow \frac{\partial L[\phi_t]}{\partial \phi}^2$$

- Normalize:

$$\phi_{t+1} \leftarrow \phi_t - \alpha \cdot \frac{\mathbf{m}_{t+1}}{\sqrt{\mathbf{v}_{t+1}} + \epsilon}$$

Small +ve constant

Normalized gradients

- Measure mean and pointwise squared gradient

$$\mathbf{m}_{t+1} \leftarrow \frac{\partial L[\boldsymbol{\phi}_t]}{\partial \boldsymbol{\phi}}$$

$$\mathbf{v}_{t+1} \leftarrow \frac{\partial L[\boldsymbol{\phi}_t]}{\partial \boldsymbol{\phi}}^2$$

$$\mathbf{m}_{t+1} = \begin{bmatrix} 3.0 \\ -2.0 \\ 5.0 \end{bmatrix}$$

$$\mathbf{v}_{t+1} = \begin{bmatrix} 9.0 \\ 4.0 \\ 25.0 \end{bmatrix}$$

- Normalize:

$$\boldsymbol{\phi}_{t+1} \leftarrow \boldsymbol{\phi}_t - \alpha \cdot \frac{\mathbf{m}_{t+1}}{\sqrt{\mathbf{v}_{t+1}} + \epsilon}$$

$$\frac{\mathbf{m}_{t+1}}{\sqrt{\mathbf{v}_{t+1}} + \epsilon} = \begin{bmatrix} 1.0 \\ -1.0 \\ 1.0 \end{bmatrix}$$

Normalized gradients

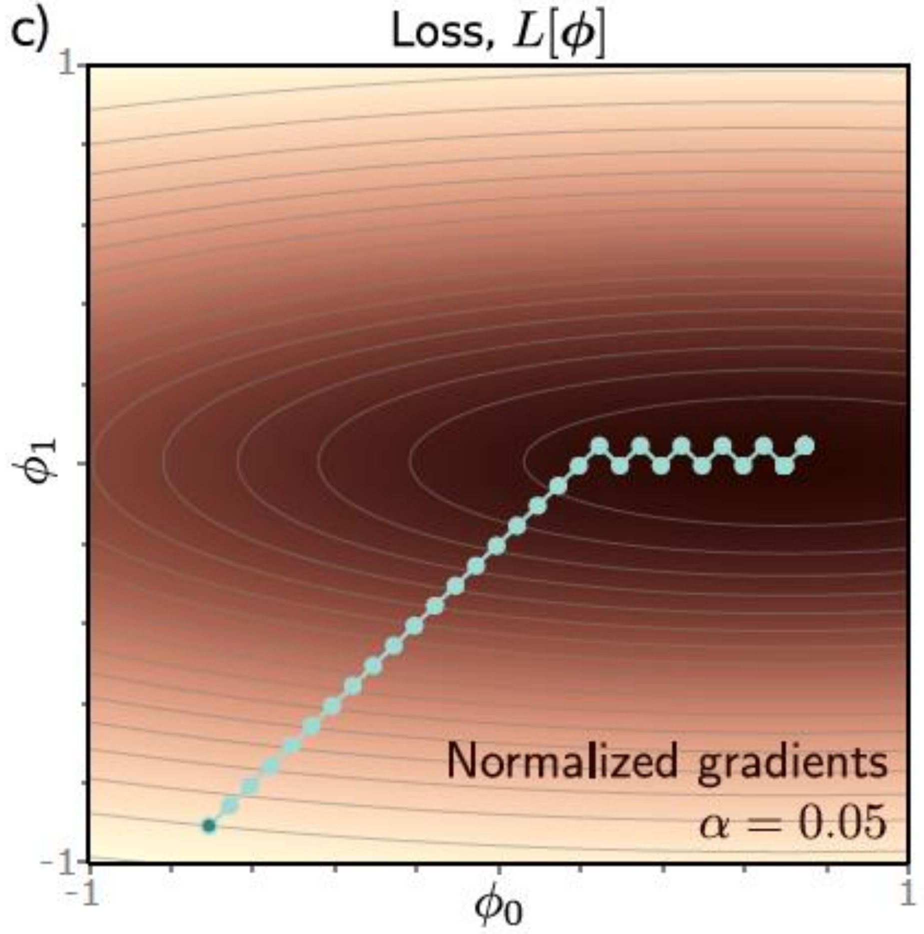
- Measure mean and pointwise squared gradient

$$\mathbf{m}_{t+1} \leftarrow \frac{\partial L[\boldsymbol{\phi}_t]}{\partial \boldsymbol{\phi}}$$

$$\mathbf{v}_{t+1} \leftarrow \frac{\partial L[\boldsymbol{\phi}_t]}{\partial \boldsymbol{\phi}}^2$$

- Normalize:

$$\boldsymbol{\phi}_{t+1} \leftarrow \boldsymbol{\phi}_t - \alpha \cdot \frac{\mathbf{m}_{t+1}}{\sqrt{\mathbf{v}_{t+1}} + \epsilon}$$



Problem with this approach: Makes good progress, but may not converge

Adaptive moment estimation (Adam)

- Compute mean and pointwise squared gradients with momentum

$$\mathbf{m}_{t+1} \leftarrow \beta \cdot \mathbf{m}_t + (1-\beta)\frac{\partial L[\boldsymbol{\phi}_t]}{\partial \boldsymbol{\phi}}$$

$$\mathbf{v}_{t+1} \leftarrow \gamma \cdot \mathbf{v}_t + (1-\gamma)\left(\frac{\partial L[\boldsymbol{\phi}_t]}{\partial \boldsymbol{\phi}}\right)^2$$

RMSProp

- Moderate near start of the sequence

$$\tilde{\mathbf{m}}_{t+1} \leftarrow \frac{\mathbf{m}_{t+1}}{1-\beta^{t+1}}$$

$$\tilde{\mathbf{v}}_{t+1} \leftarrow \frac{\mathbf{v}_{t+1}}{1-\gamma^{t+1}}$$

- Update the parameters

$$\boxed{\boldsymbol{\phi}_{t+1} \leftarrow \boldsymbol{\phi}_t - \alpha \cdot \frac{\tilde{\mathbf{m}}_{t+1}}{\sqrt{\tilde{\mathbf{v}}_{t+1}}+\epsilon}}$$

$$m_0 = 0$$

$$m_1 = \beta \cdot m_0 + (1-\beta)\,\Delta L_t$$

$$T. \quad = (1-\beta)\,\Delta L_t$$

$$\boxed{\tilde{m}_1 = \frac{m_1}{1-\beta}}$$

$$m_2 = \beta\, m_1 + (1-\beta)\,\Delta L_{t+1}$$

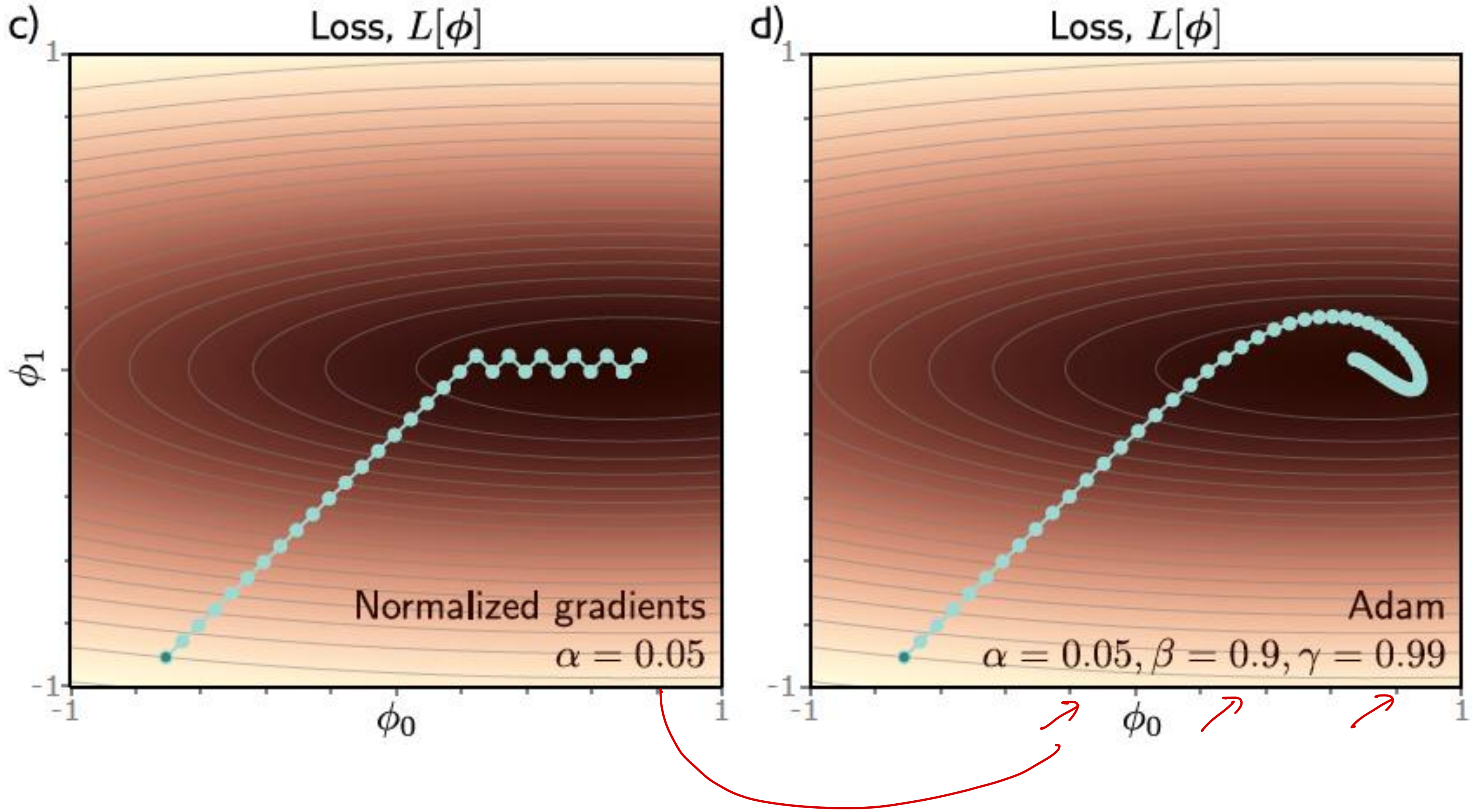
$$= \beta(1-\beta)\,\Delta L_t + (1-\beta)\,\Delta L_{t+1}$$

$$\approx \Delta L\,[\beta(1-\beta) + 1-\beta]$$

$$= \Delta L\,(1-\beta^2)$$

$$\boxed{\tilde{m}_2 = \frac{m_2}{1-\beta^2}}$$

Adaptive moment estimation (Adam)



Hyperparameters

- Choice of learning algorithm
- Learning rate
- Momentum

SGD – Batch size

Hyperparameters

$$\boxed{-\sum_{i=1}^{I} \log Pr(y_i | x_i)}$$

Consider building a model to predict the number of pedestrians $y \in \{0, 1, 2, \ldots, \}$ that will pass a given point in the city in the next minute, based on data x that contains information about the time of day, the longitude and latitude, and the type of neighborhood. A suitable distribution for modeling counts is the Poisson distribution. This has a single parameter $\lambda > 0$ called the rate that represents the mean of the distribution. The distribution has probability density function:

$$Pr(y = k) = \frac{\lambda^k e^{-\lambda}}{k!}$$

Design a loss function for this model assuming that we have access to I training pairs $\{x_i, y_i\}$.