

# Deep Learning for Data Science

## DS 542

<https://dl4ds.github.io/sp2026/>

Fitting Models

# Plan for Today

- Homework 3 post-mortem
- Gradient descent review
- Stochastic gradient descent (more formally)
- Momentum
- Adam

# Homework 3 Post-Mortem

Raise your hand if you encountered any of the following.

- Bad prediction accuracy
- Loss function improving very slowly
- Loss function going up
- NaN or infinity in loss calculations
- NaN in initial loss calculations?

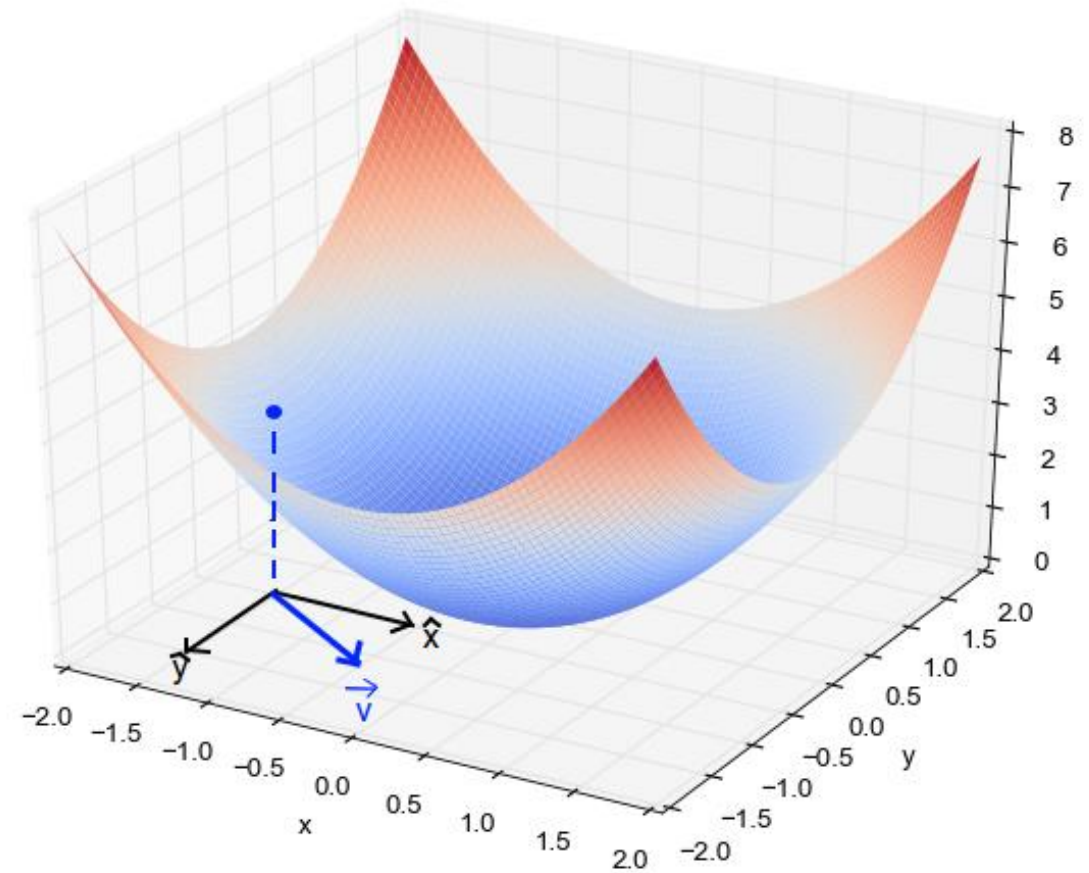
# Plan for Today

- Homework 3 post-mortem
- Gradient descent review
- Stochastic gradient descent (more formally)
- Momentum
- Adam

# Gradient

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

Partial derivative, e.g. rate of change, w.r.t. each input (independent) variable.



Geometric Interpretation: Each variable is a unit vector, and then

- gradient is the rate of change (increase) in the direction of each unit vector
- vector sum points to the overall direction of greatest change (increase)

# Gradient descent algorithm

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

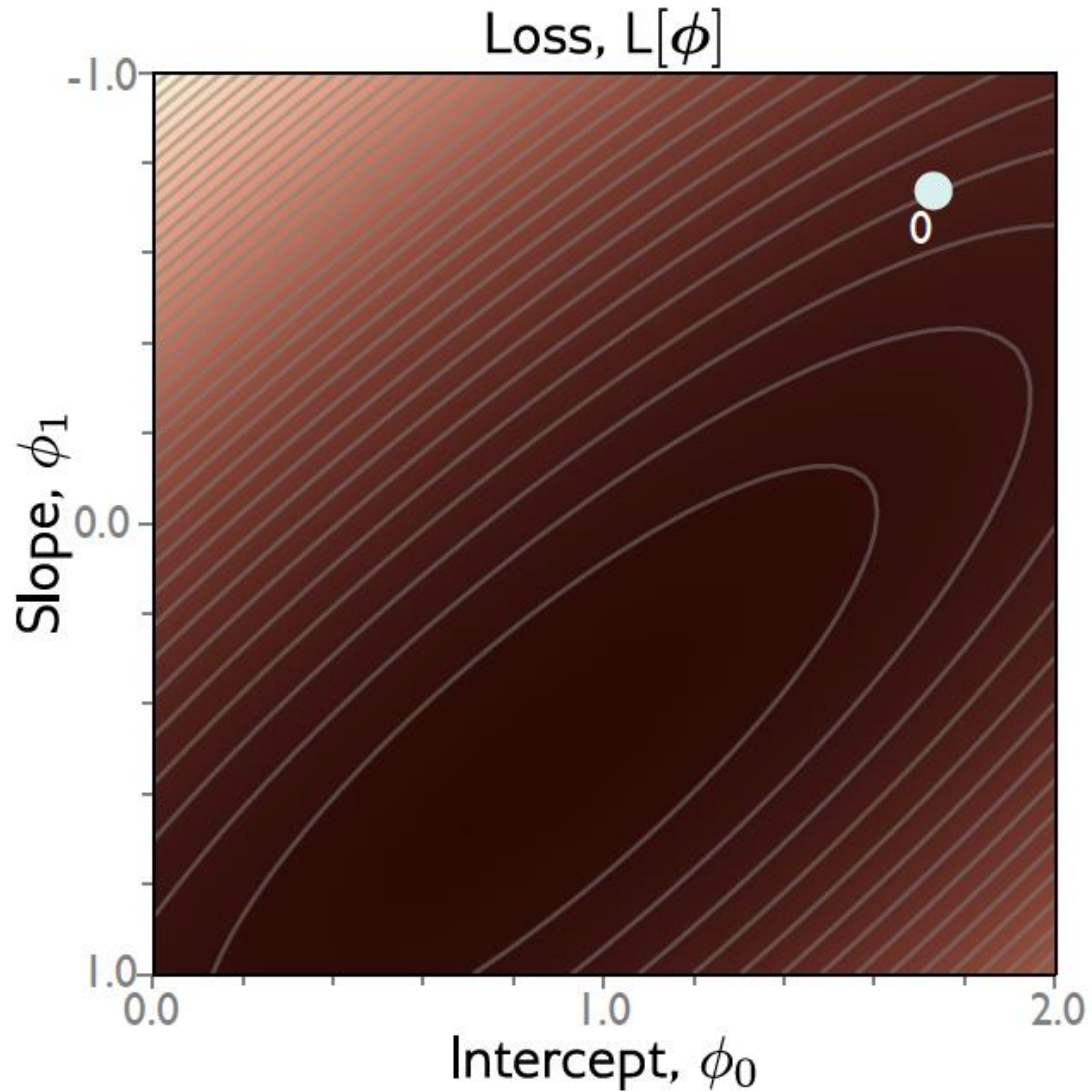
$$\frac{\partial L}{\partial \phi} = \begin{bmatrix} \frac{\partial L}{\partial \phi_0} \\ \frac{\partial L}{\partial \phi_1} \\ \vdots \\ \frac{\partial L}{\partial \phi_N} \end{bmatrix}. \quad \text{Also notated as } \nabla_w L$$

**Step 2.** Update the parameters according to the rule:

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

where the positive scalar  $\alpha$  determines the magnitude of the change.

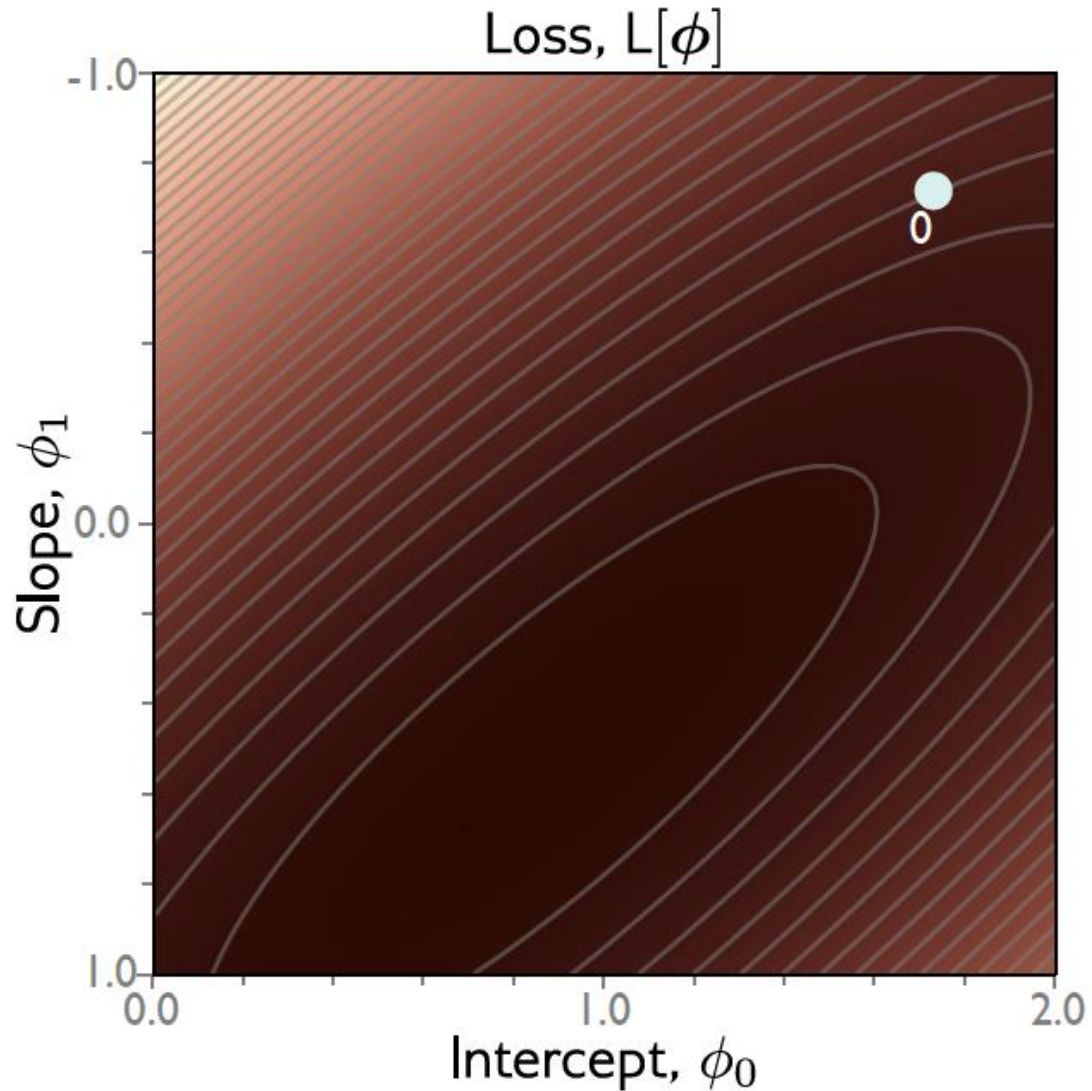
# Gradient descent



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

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

# Gradient descent



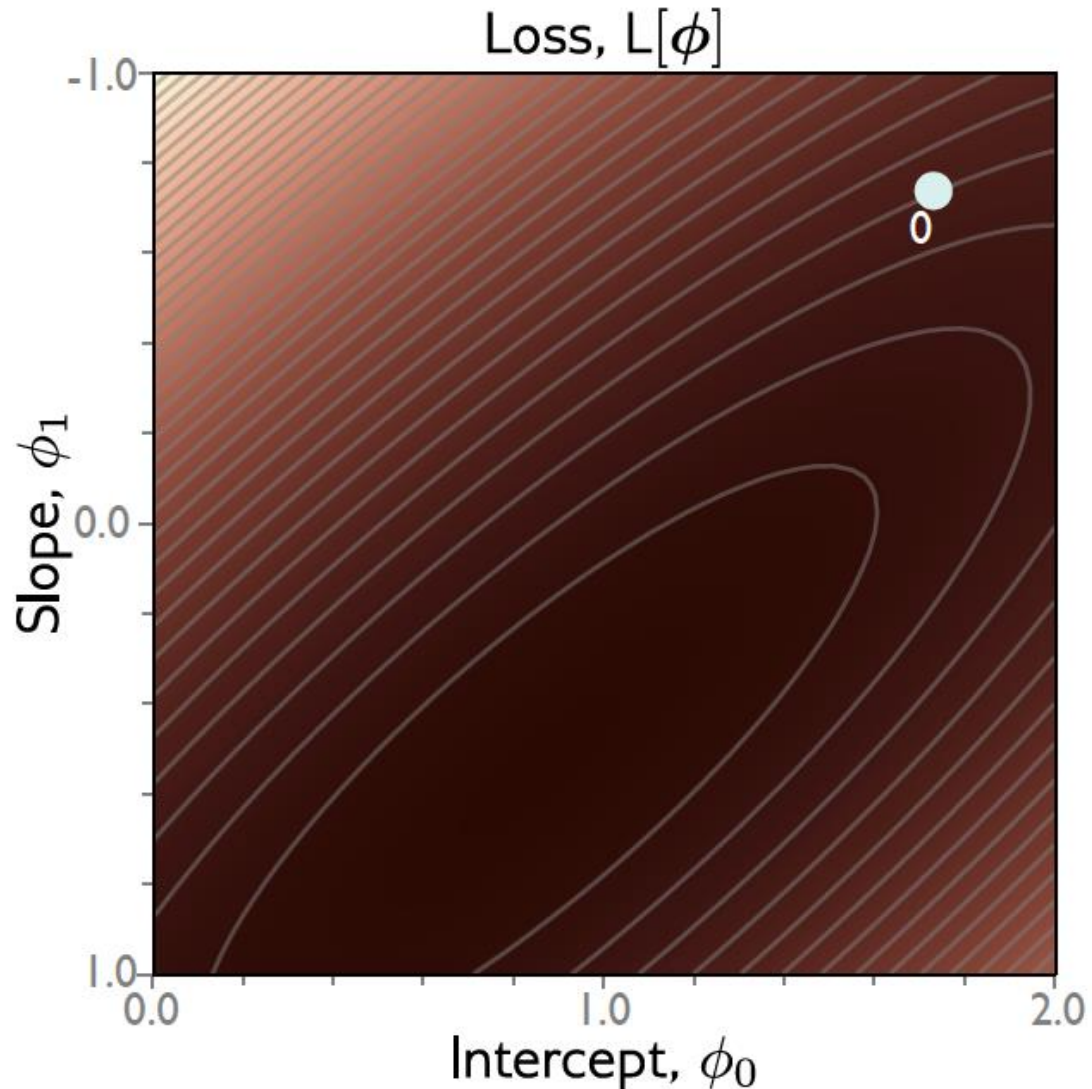
Step 1: Compute derivatives (slopes of function) with Respect to the parameters

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

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



# Gradient descent



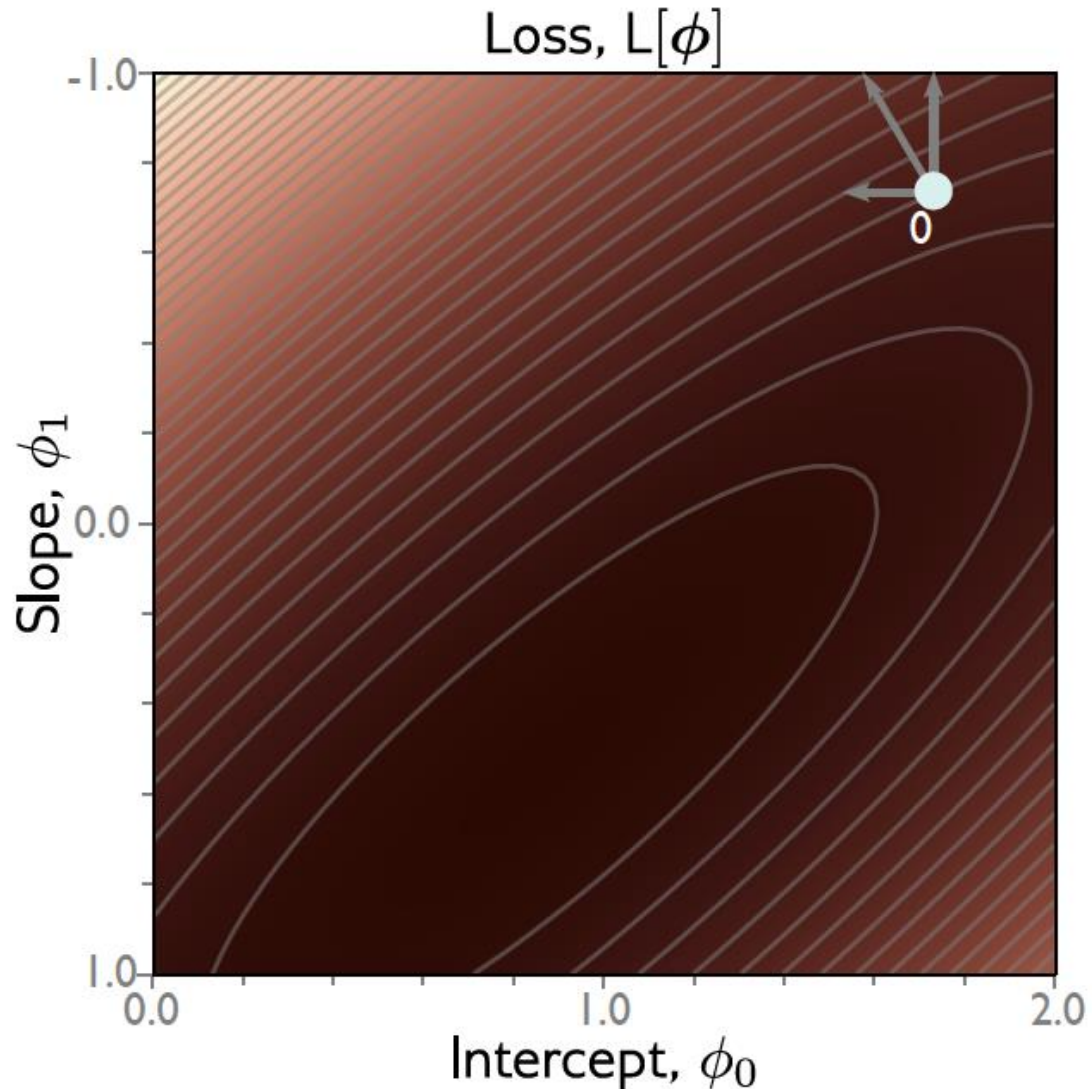
Step 1: Compute derivatives (slopes of function) with Respect to the parameters

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

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

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

# Gradient descent

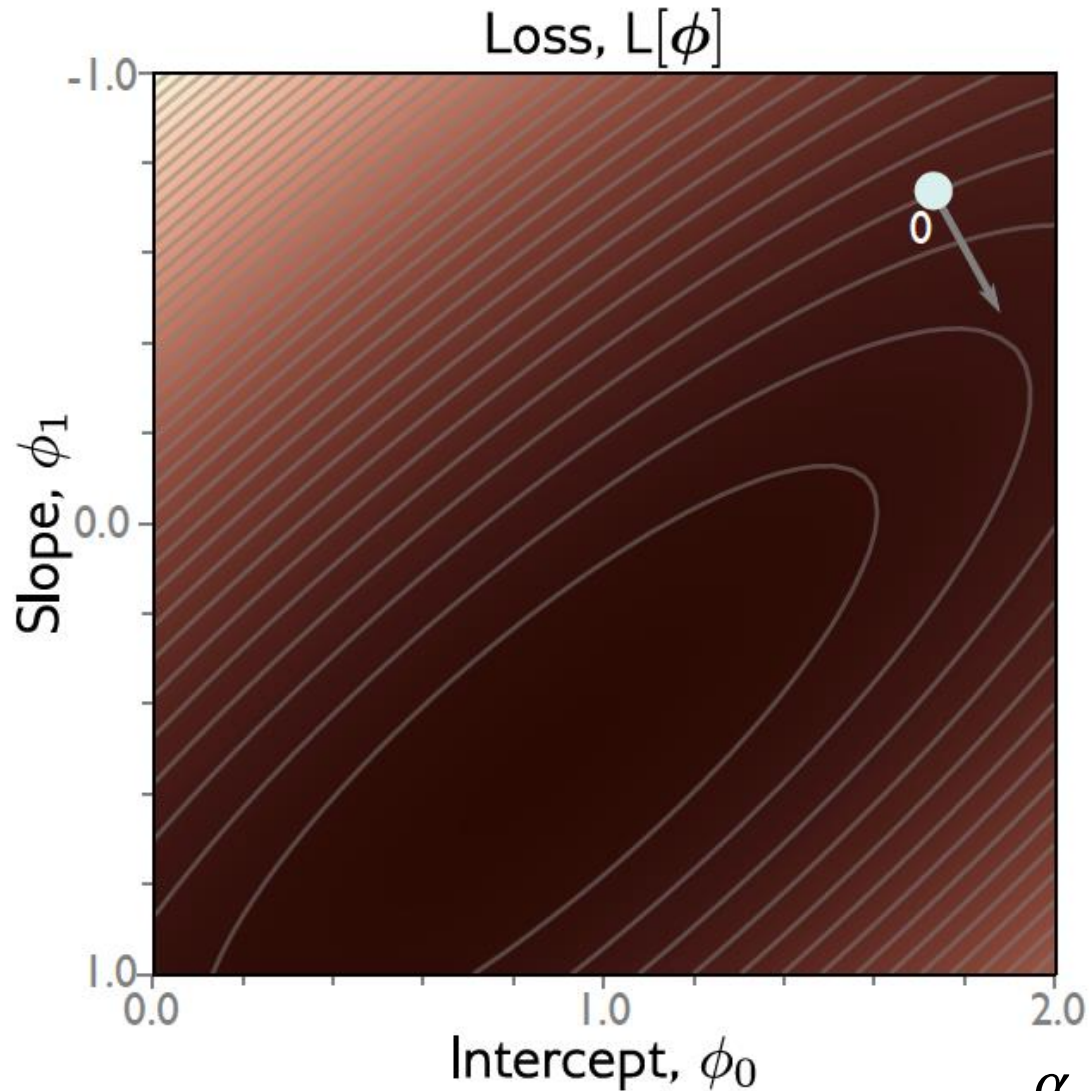


Step 1: Compute derivatives (slopes of function) with Respect to the parameters

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

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

# Gradient descent



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

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

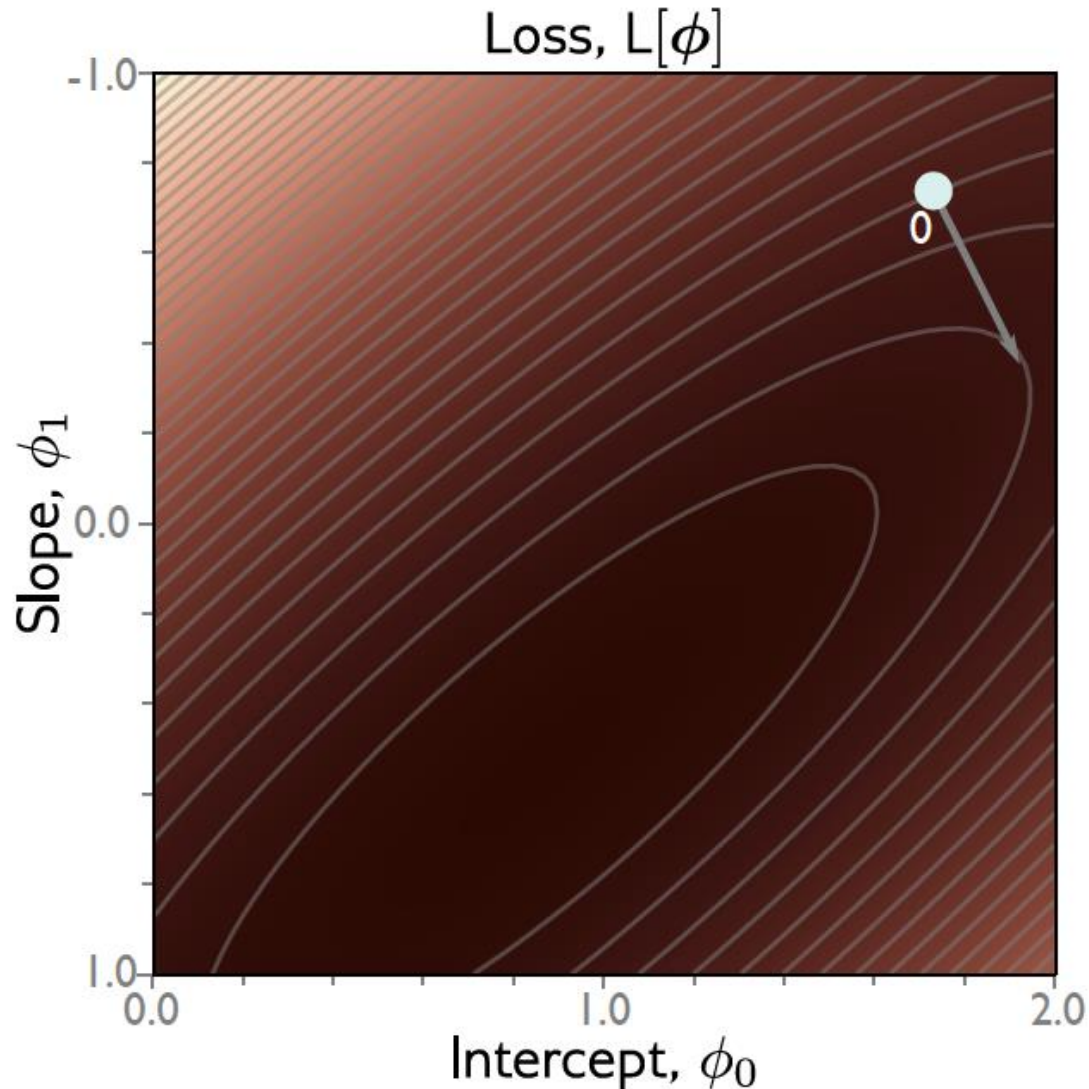
$$\frac{\partial \ell_i}{\partial \phi} = \begin{bmatrix} \frac{\partial \ell_i}{\partial \phi_0} \\ \frac{\partial \ell_i}{\partial \phi_1} \end{bmatrix} = \begin{bmatrix} 2(\phi_0 + \phi_1 x_i - y_i) \\ 2x_i(\phi_0 + \phi_1 x_i - y_i) \end{bmatrix}$$

Step 2: Update parameters according to rule

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

$\alpha$  = step size or **learning rate** if fixed

# Gradient descent



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

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

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

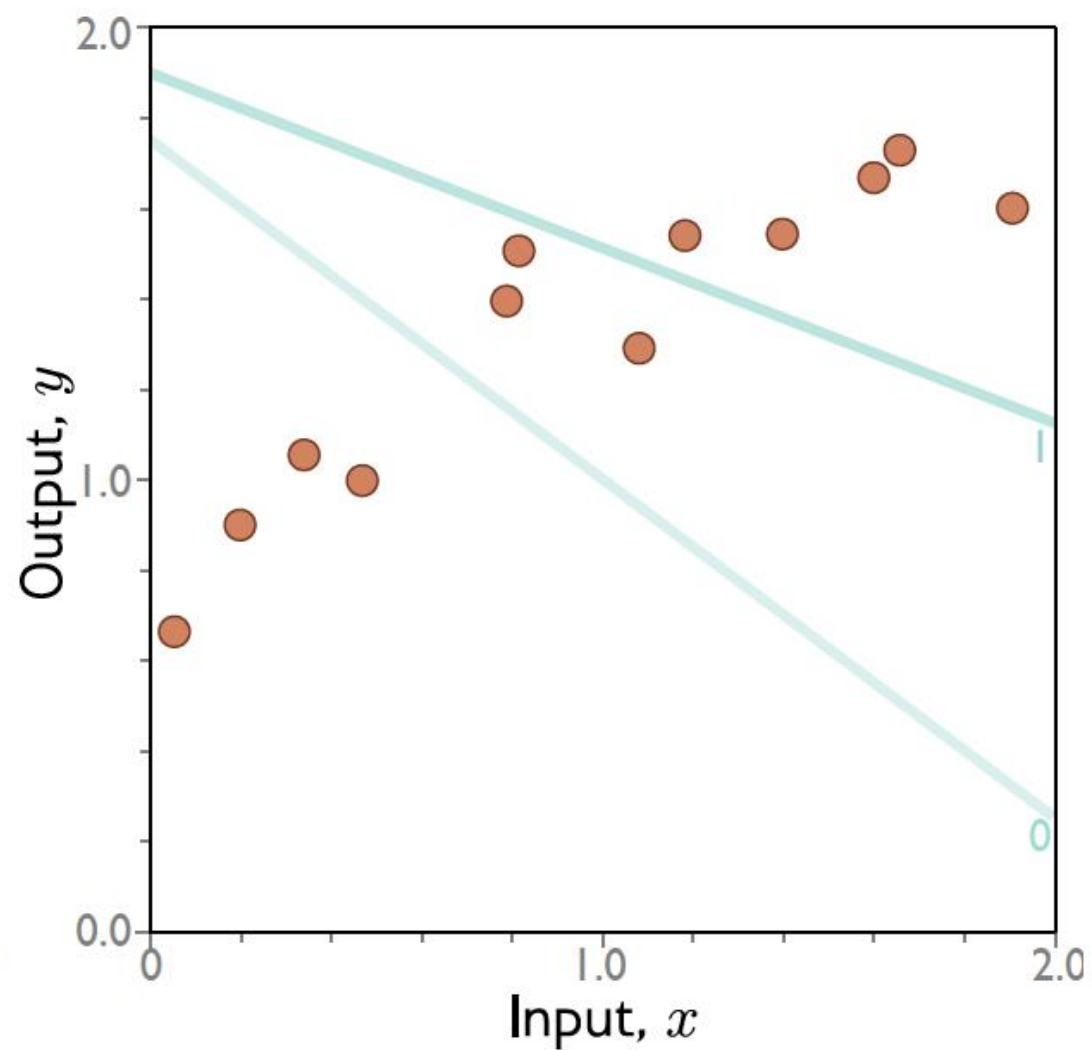
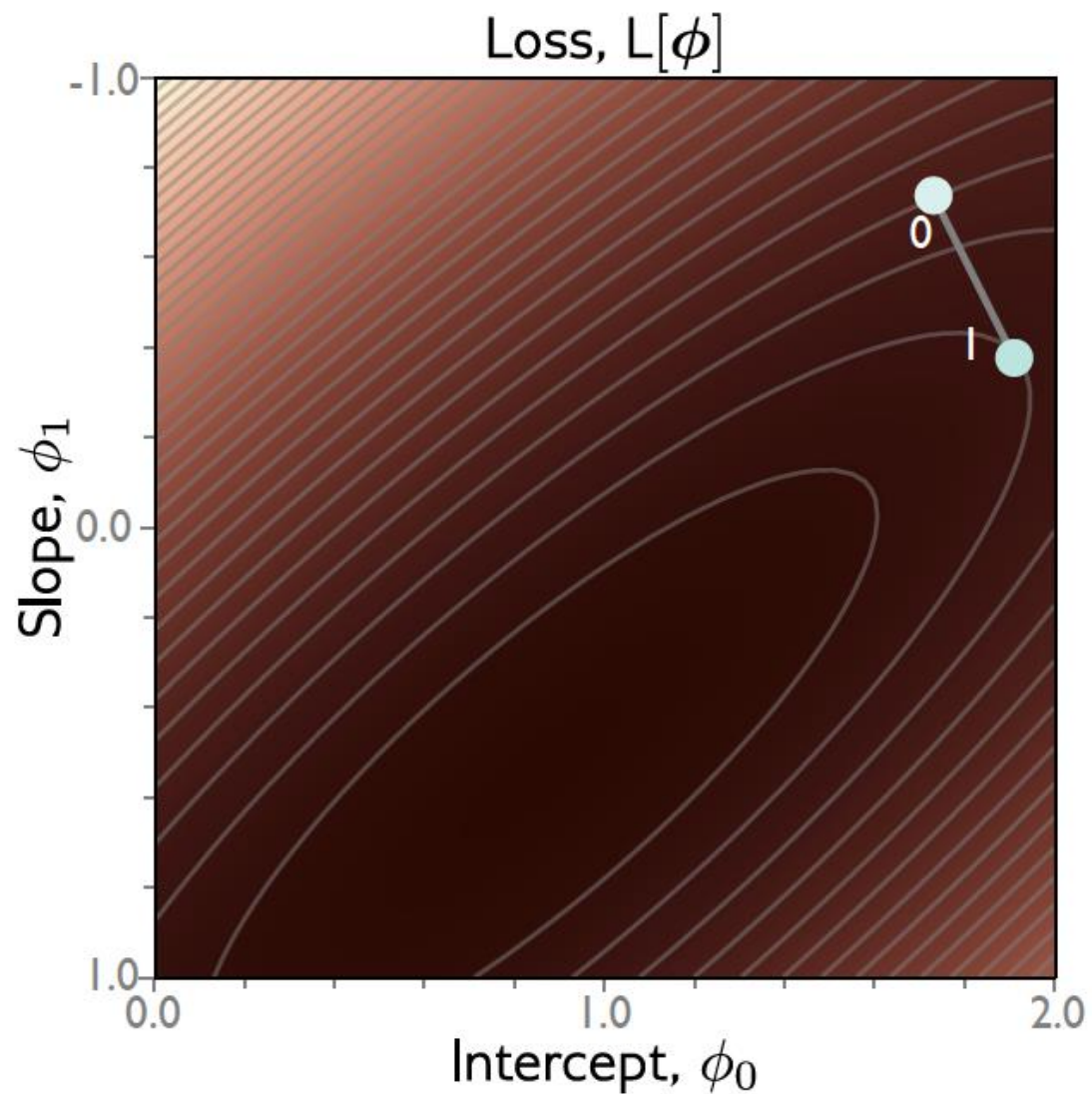
Step 2: Update parameters according to rule

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

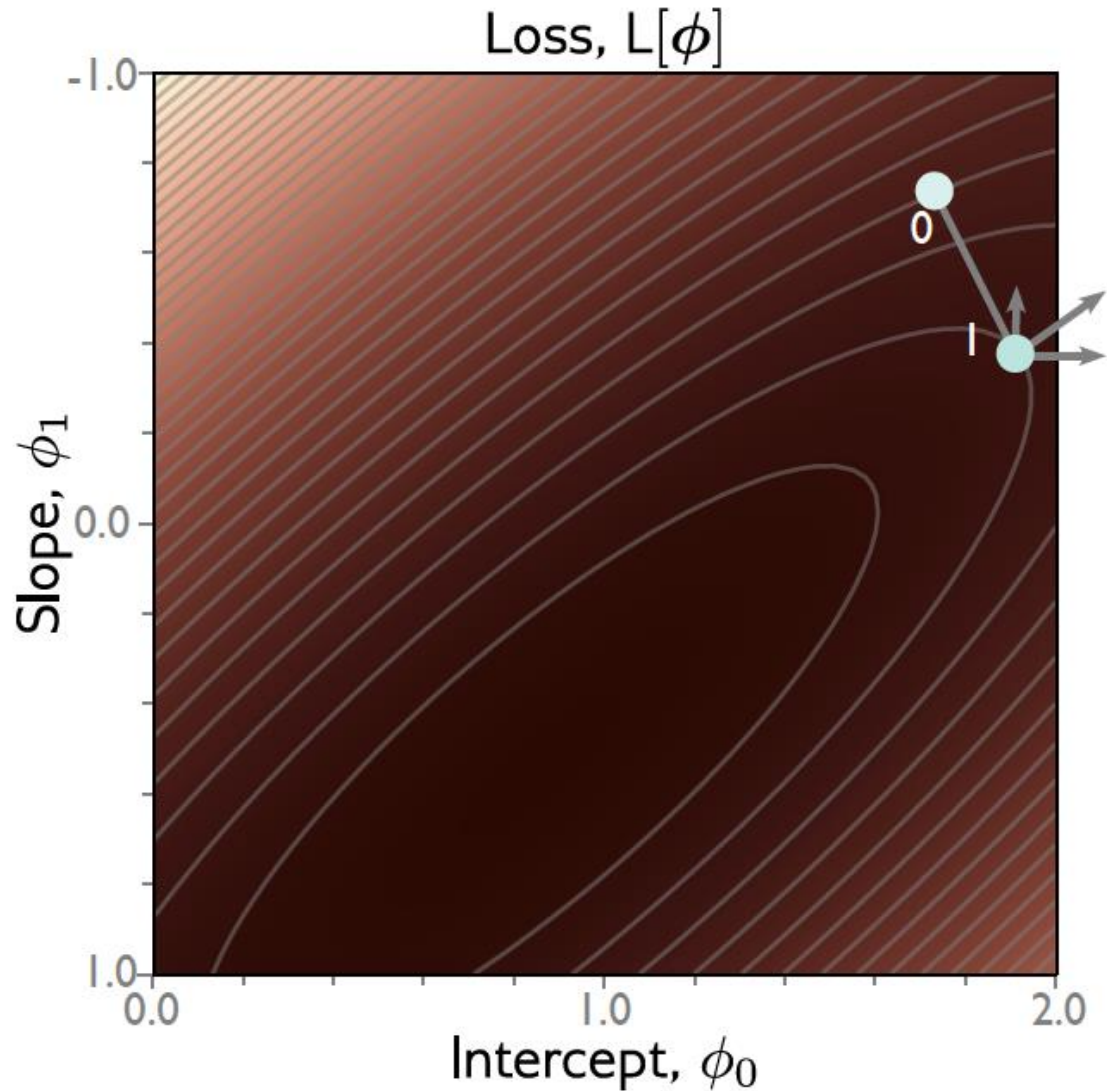
$\alpha$  = step size



# Gradient descent



# Gradient descent



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

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

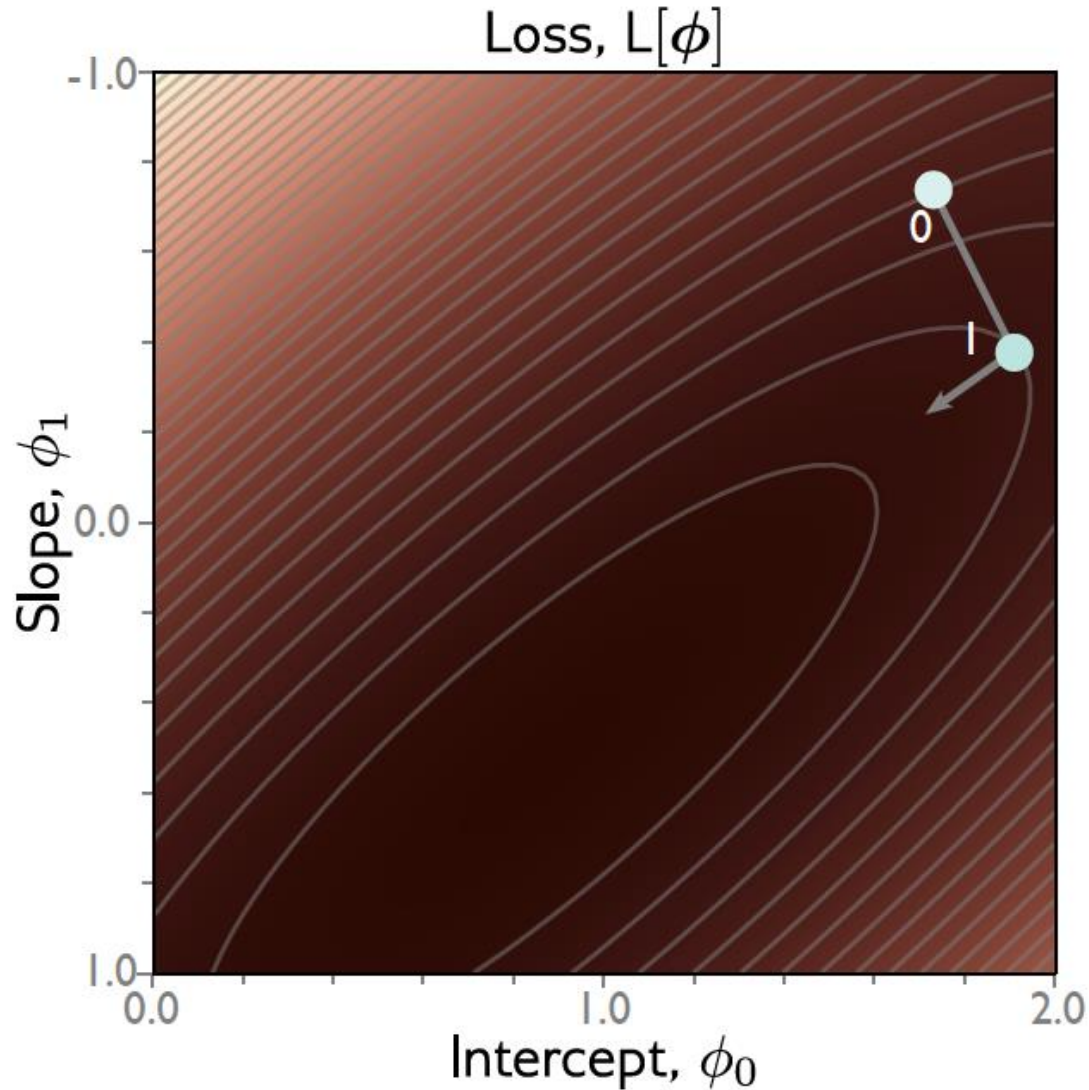
$$\frac{\partial \ell_i}{\partial \phi} = \begin{bmatrix} \frac{\partial \ell_i}{\partial \phi_0} \\ \frac{\partial \ell_i}{\partial \phi_1} \end{bmatrix} = \begin{bmatrix} 2(\phi_0 + \phi_1 x_i - y_i) \\ 2x_i(\phi_0 + \phi_1 x_i - y_i) \end{bmatrix}$$

Step 2: Update parameters according to rule

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

$\alpha$  = step size

# Gradient descent



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

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

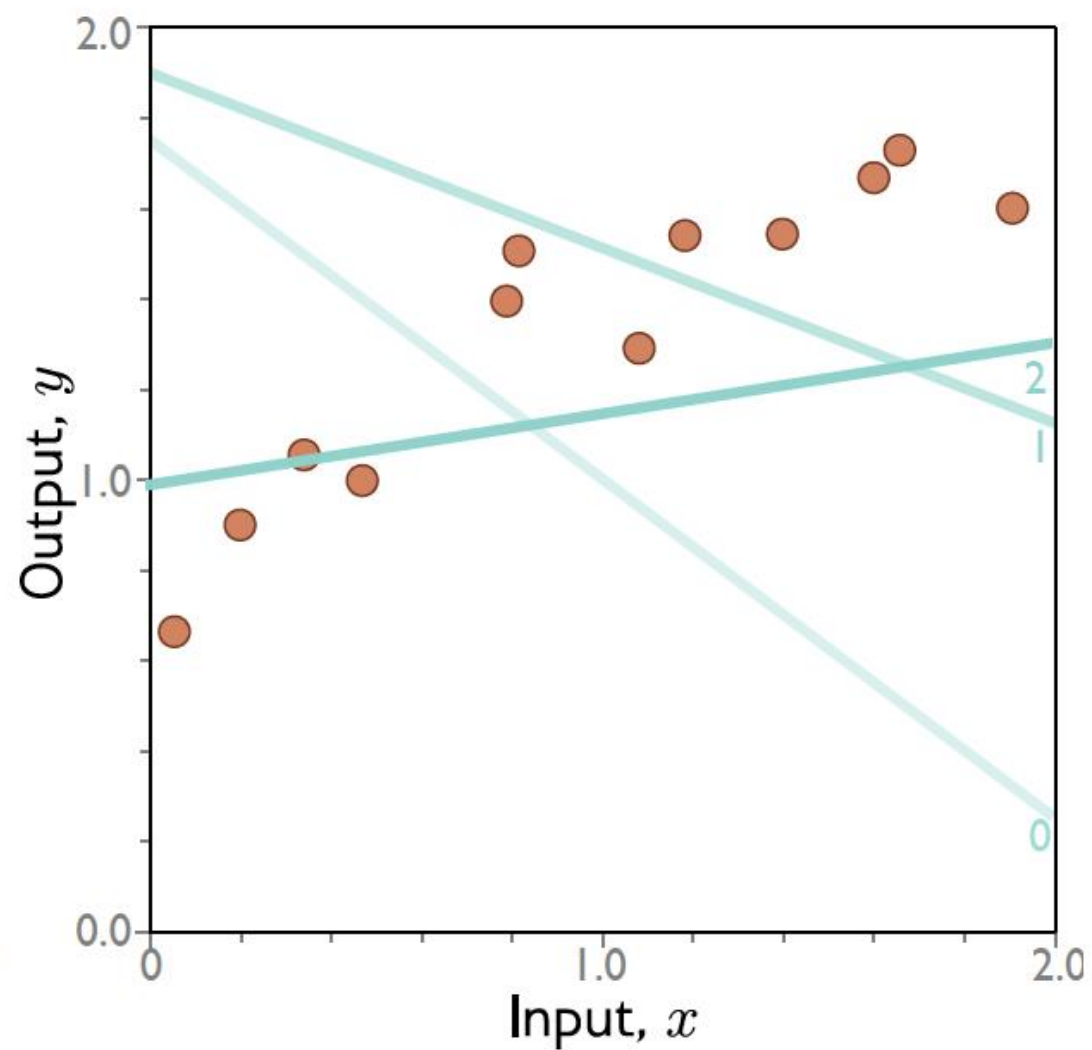
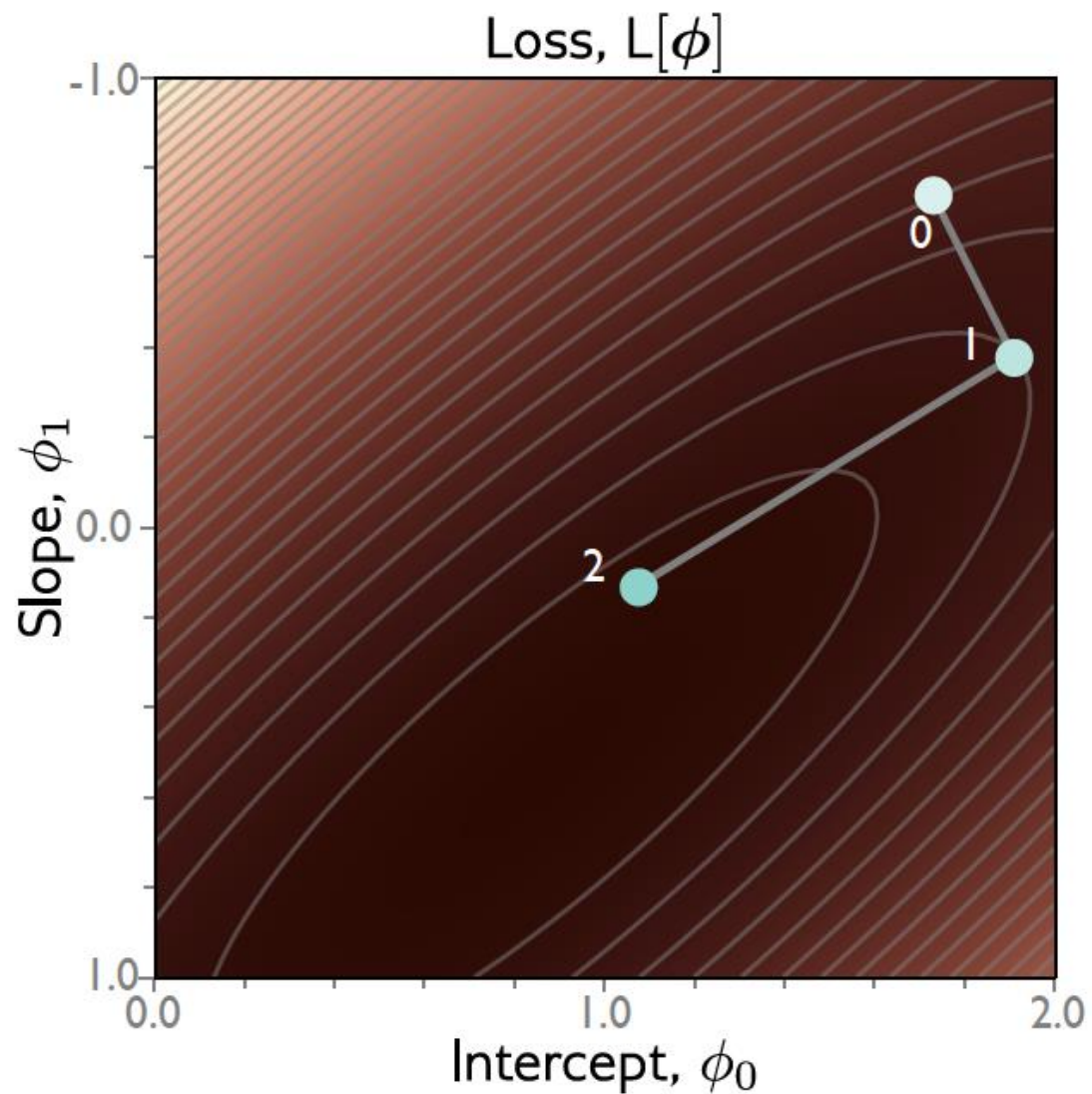
$$\frac{\partial \ell_i}{\partial \phi} = \begin{bmatrix} \frac{\partial \ell_i}{\partial \phi_0} \\ \frac{\partial \ell_i}{\partial \phi_1} \end{bmatrix} = \begin{bmatrix} 2(\phi_0 + \phi_1 x_i - y_i) \\ 2x_i(\phi_0 + \phi_1 x_i - y_i) \end{bmatrix}$$

Step 2: Update parameters according to rule

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

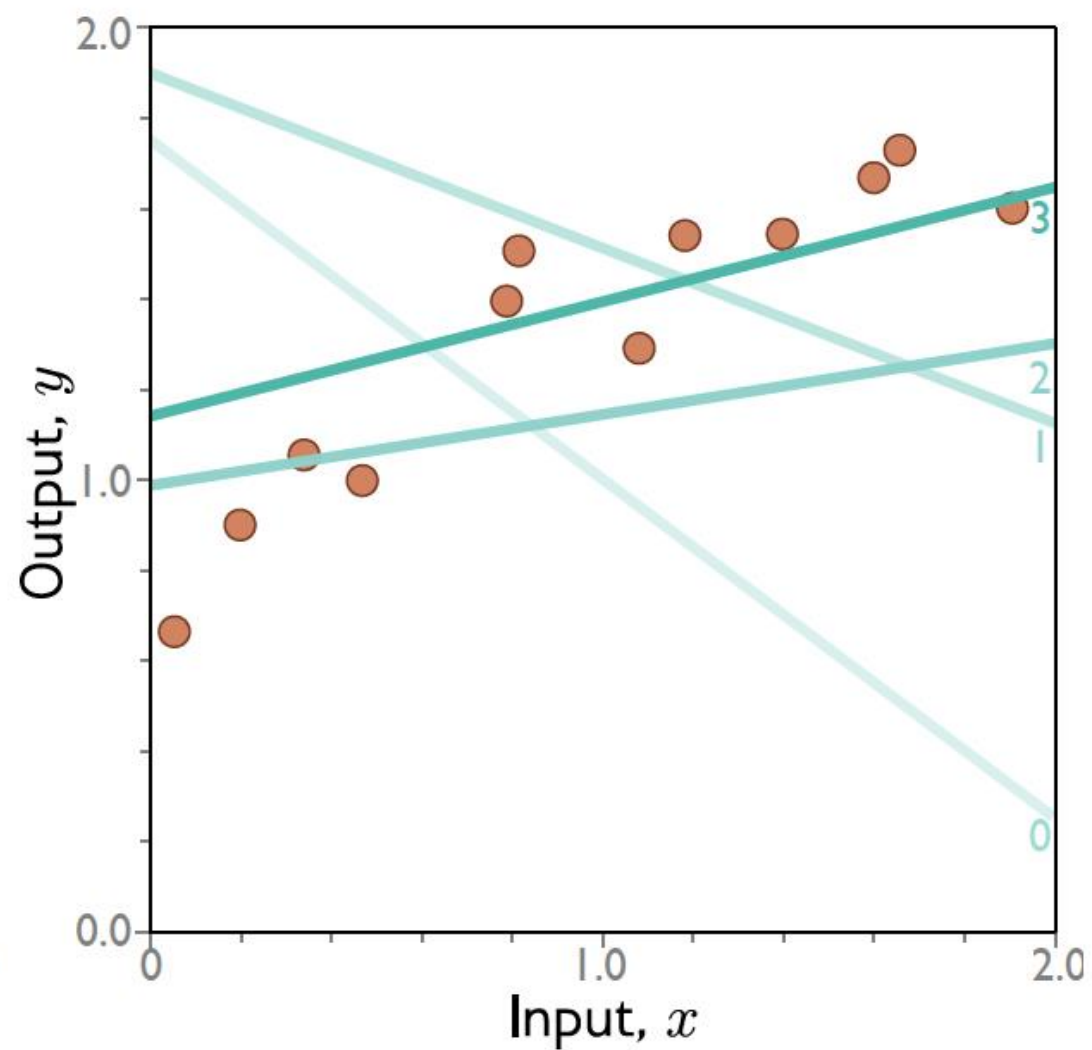
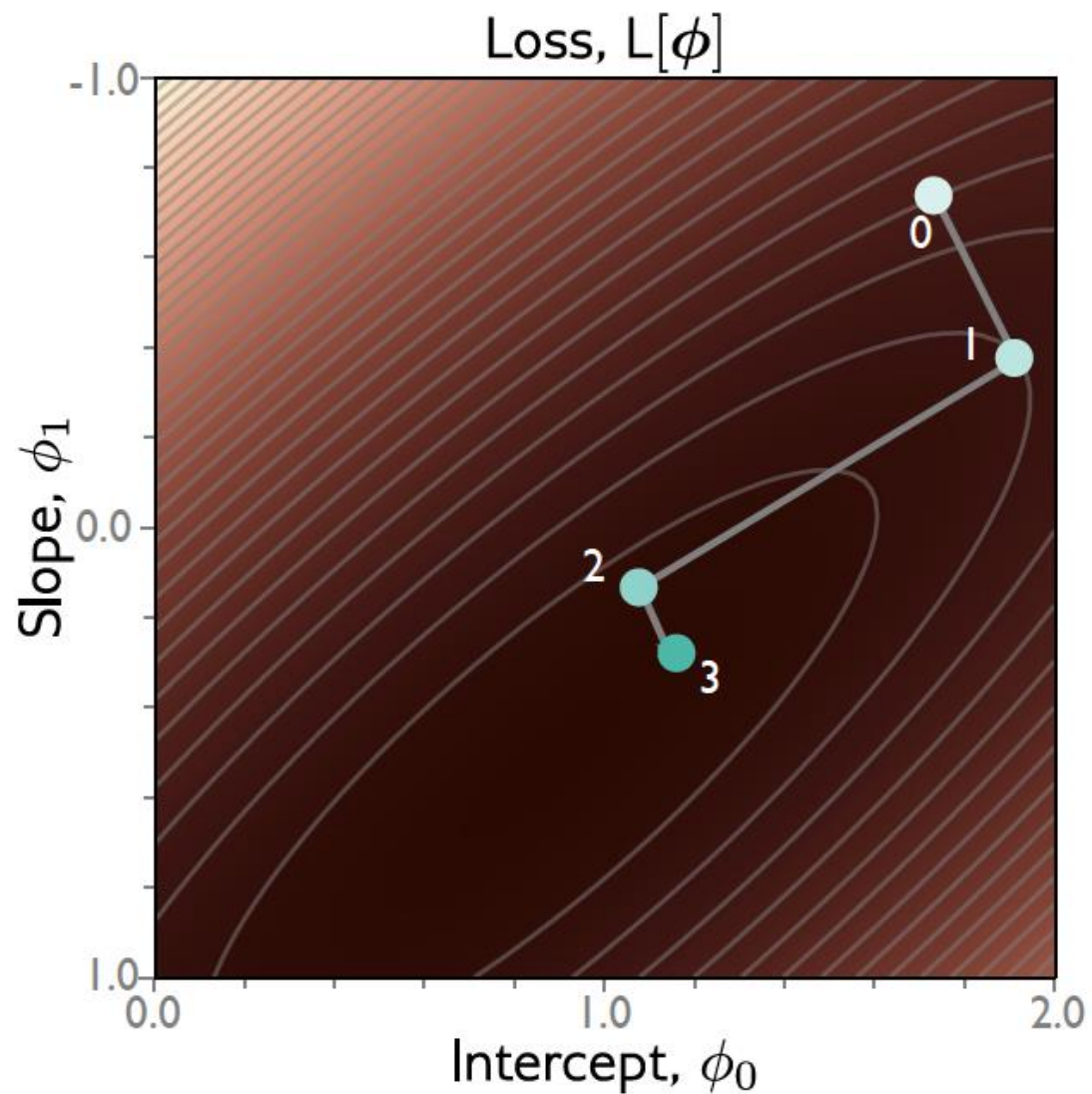
$\alpha$  = step size

# Gradient descent

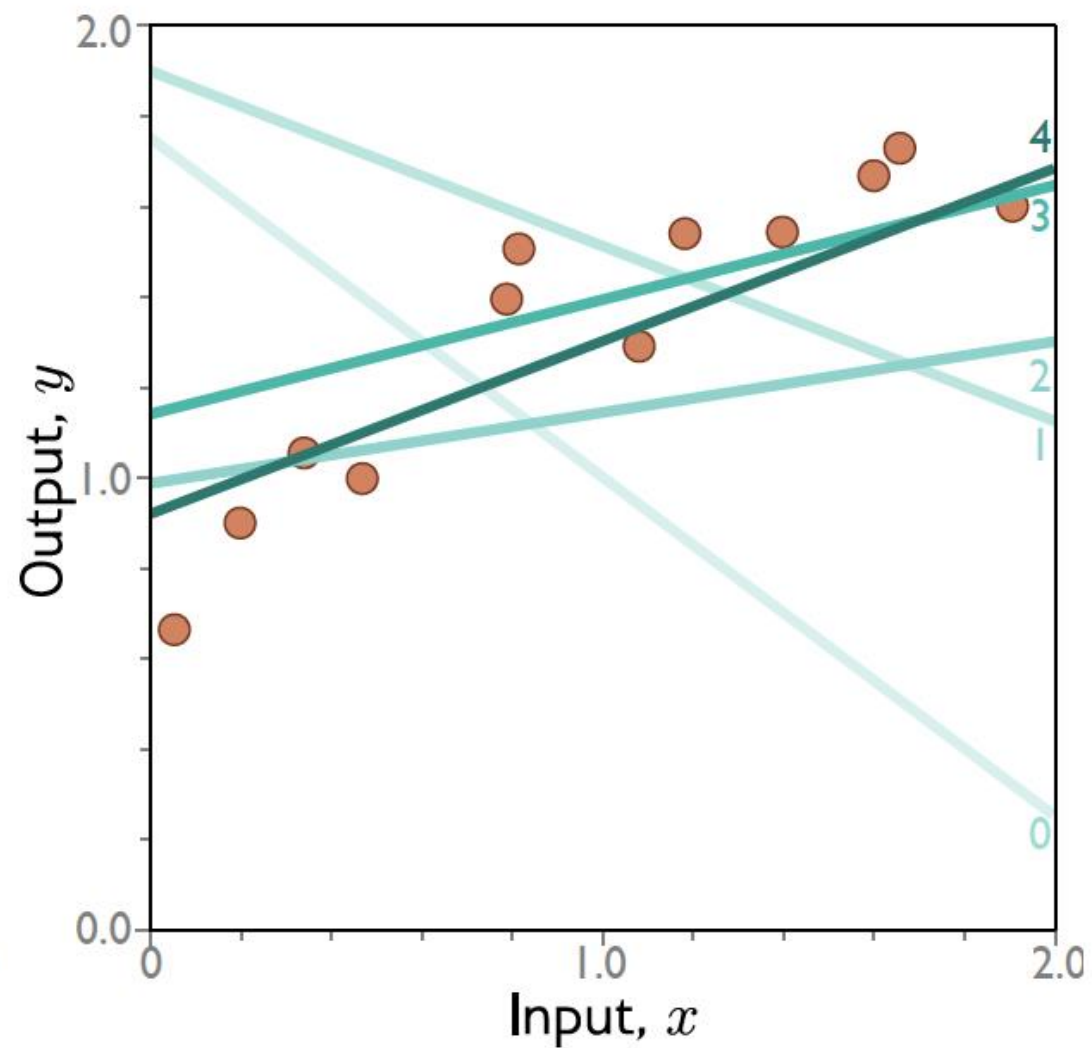
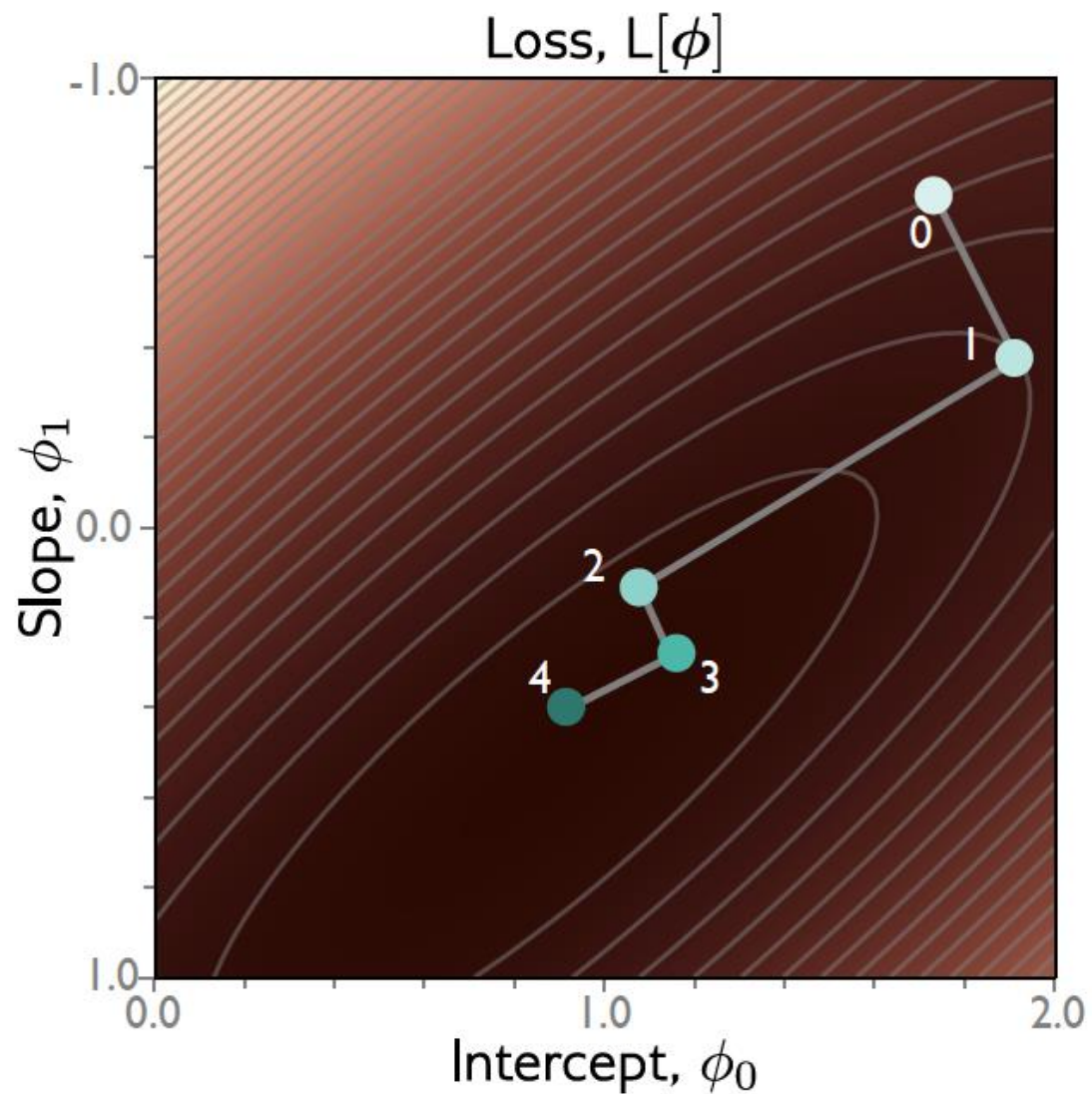




# Gradient descent



# Gradient descent



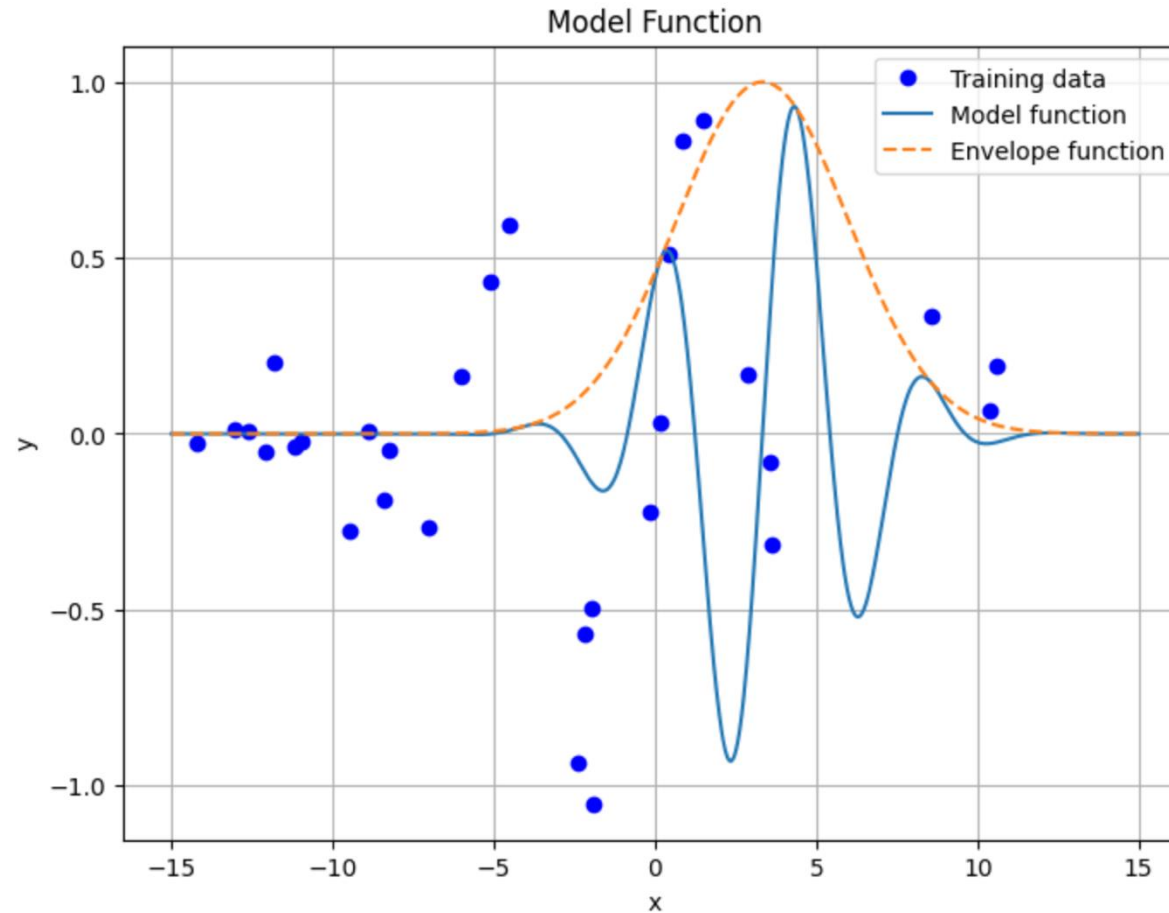
The linear model loss function was convex.

We'll use a more complex (non-convex)  
model that we can still visualize in 2D and 3D

➔ Gabor Function

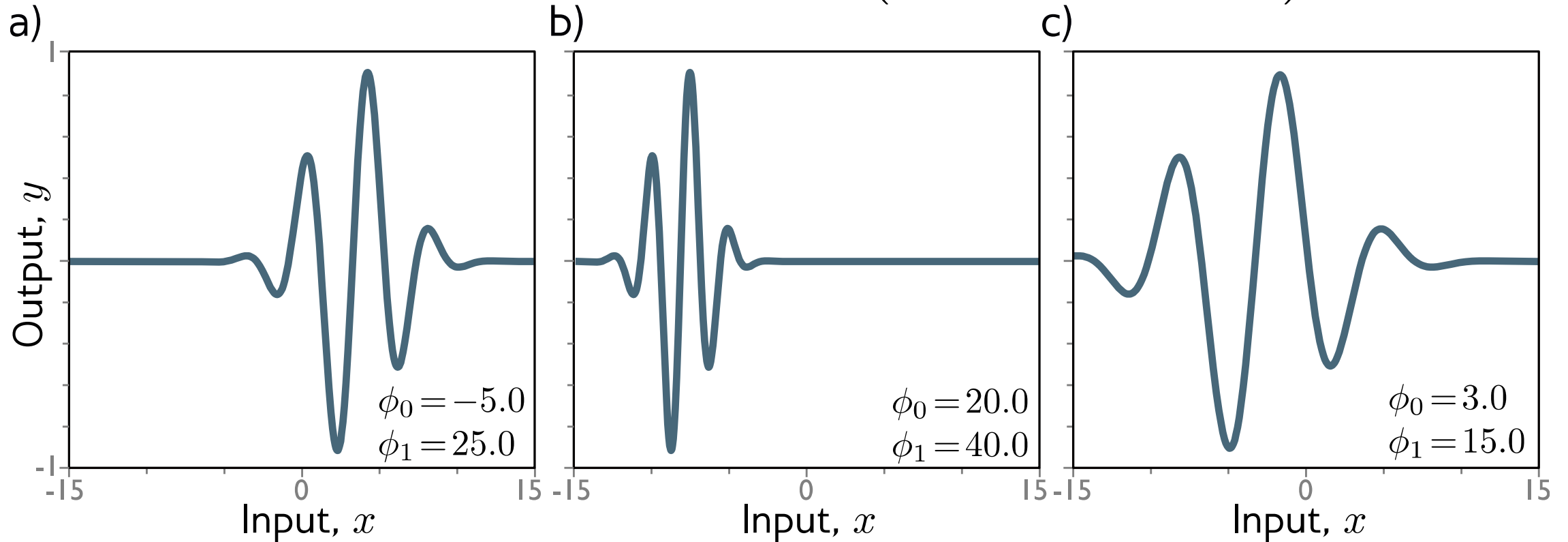
# Gabor Model (with Envelope)

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



# Gabor model

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

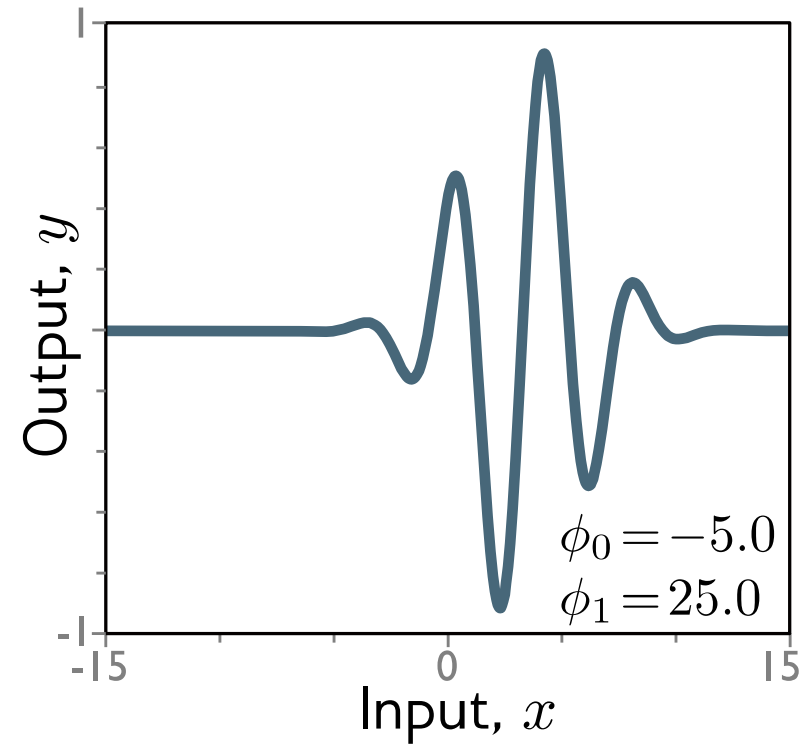
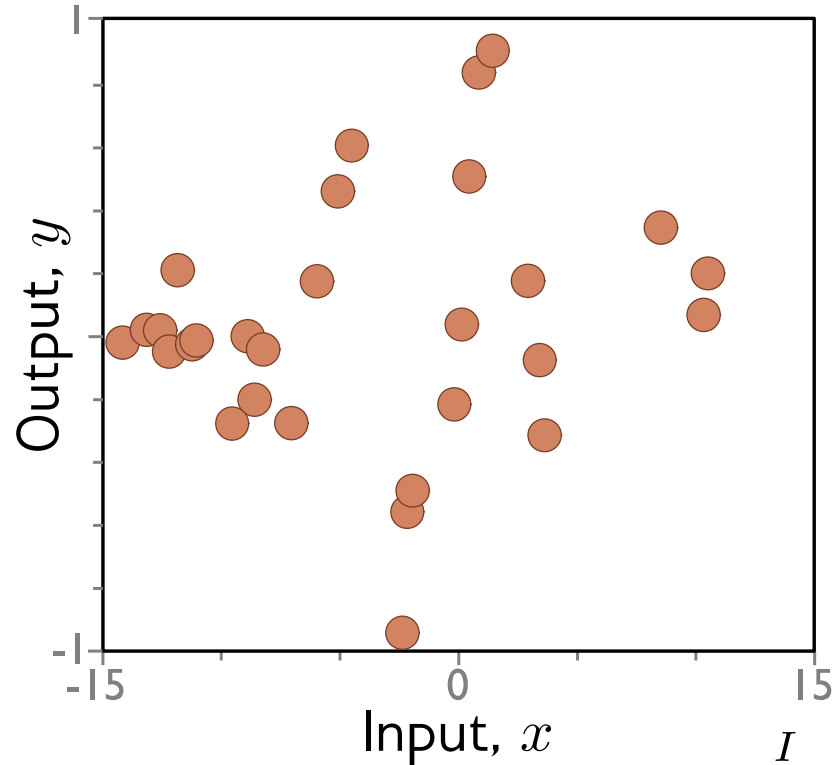


$\phi_0$  shifts left and right

$\phi_1$  shrinks and expands the sinusoid and envelope

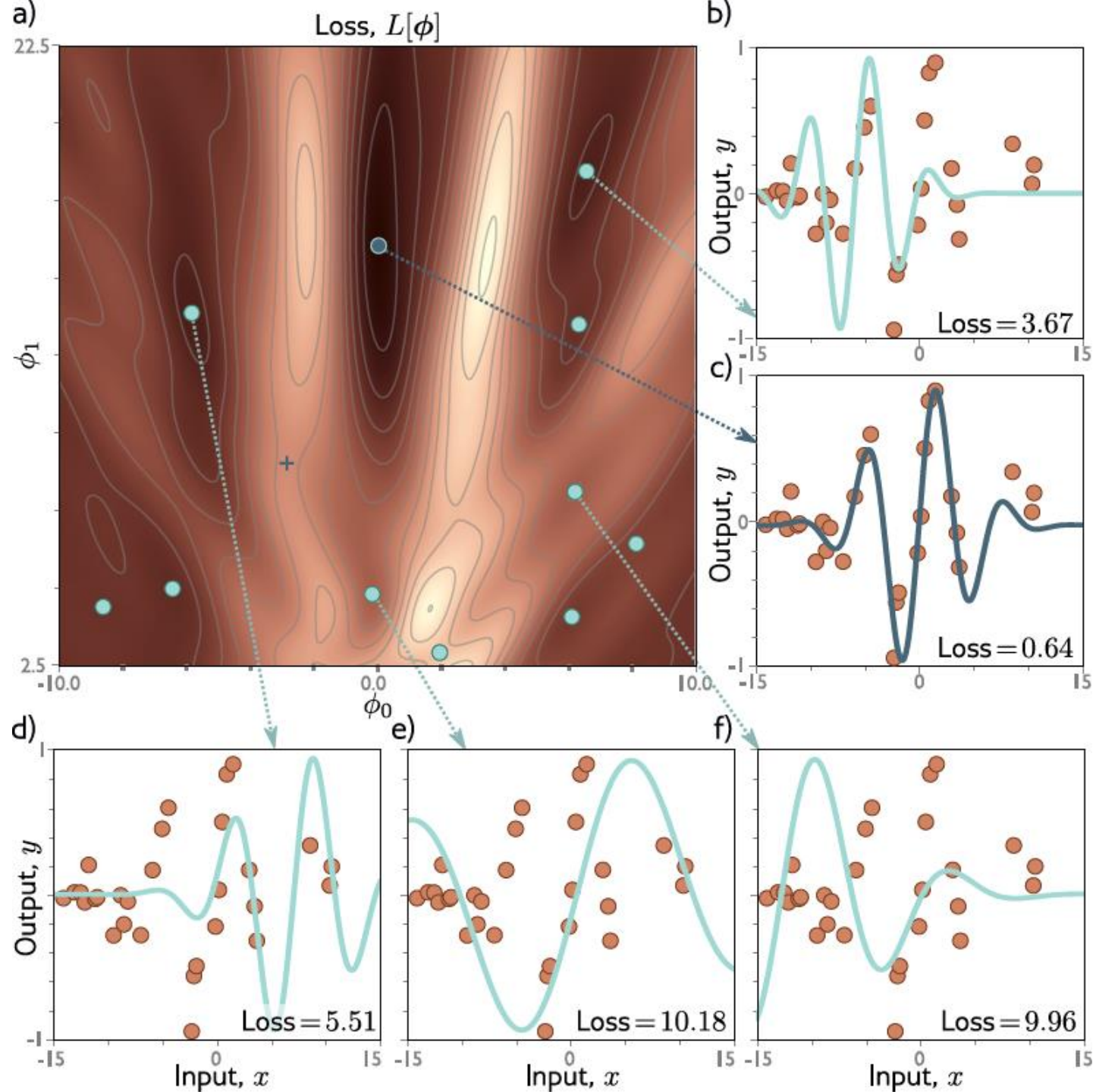
# Toy Dataset and Gabor model

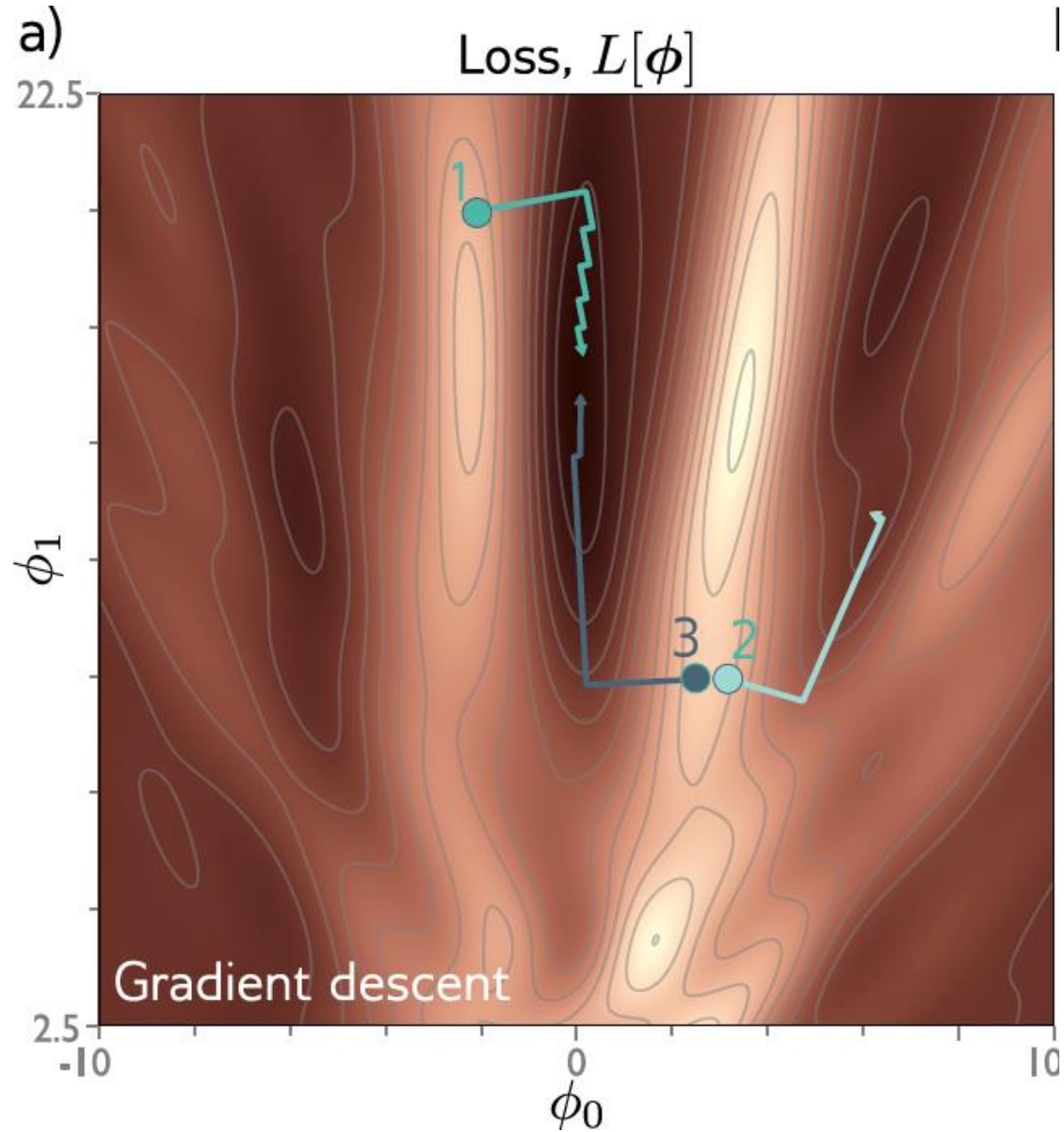
$$f[x, \phi] = \sin[\phi_0 + 0.06 \cdot \phi_1 x] \cdot \exp\left(-\frac{(\phi_0 + 0.06 \cdot \phi_1 x)^2}{8.0}\right)$$



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





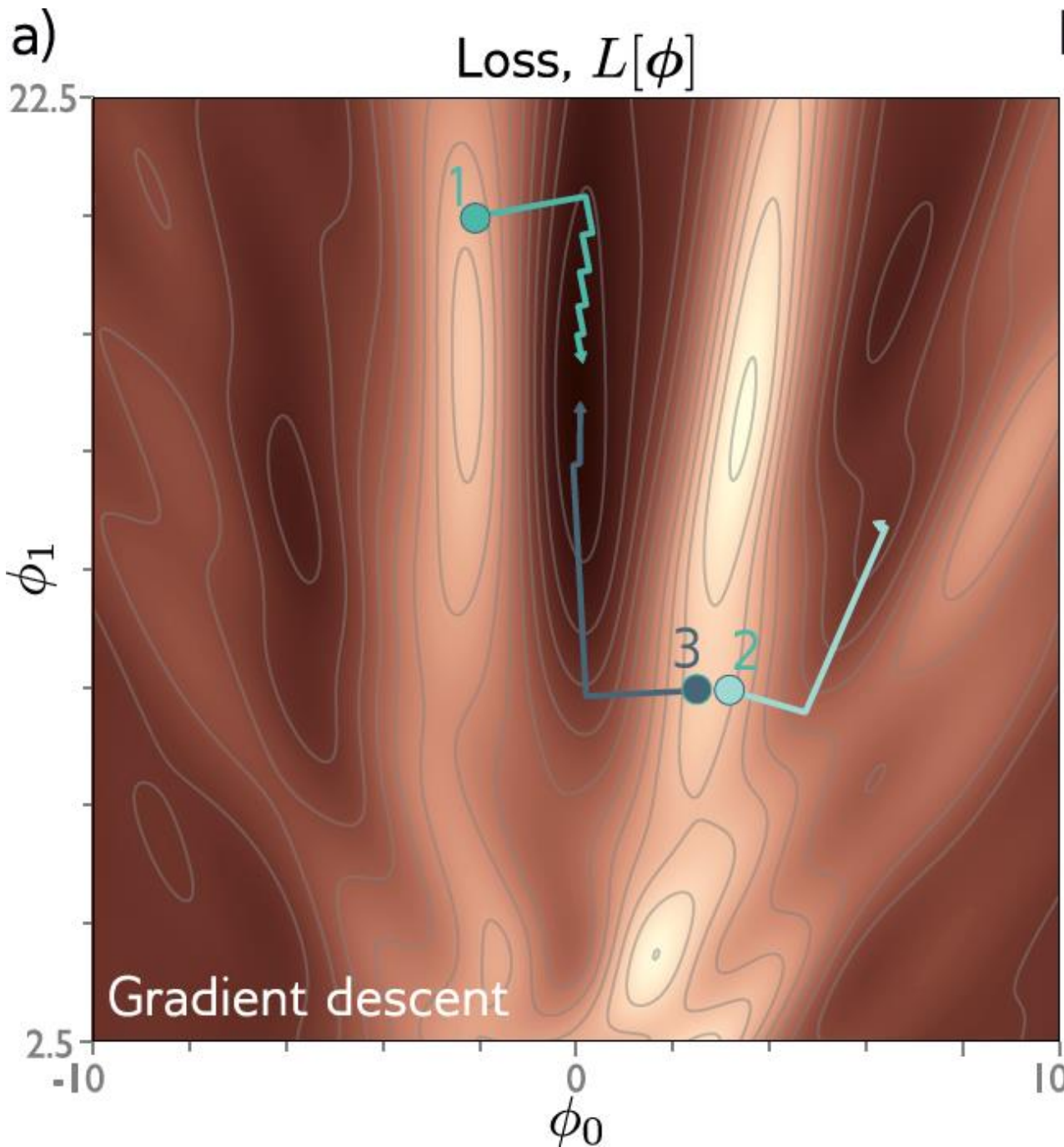


- Gradient descent gets to the global minimum if we start in the right “valley”
- Otherwise, descends to a local minimum
- Or get stuck near a saddle point



# Plan for Today

- Homework 3 post-mortem
- Gradient descent review
- Stochastic gradient descent (more formally)
- Momentum
- Adam



IDEA: add noise, save computation

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

# Batches and Epochs

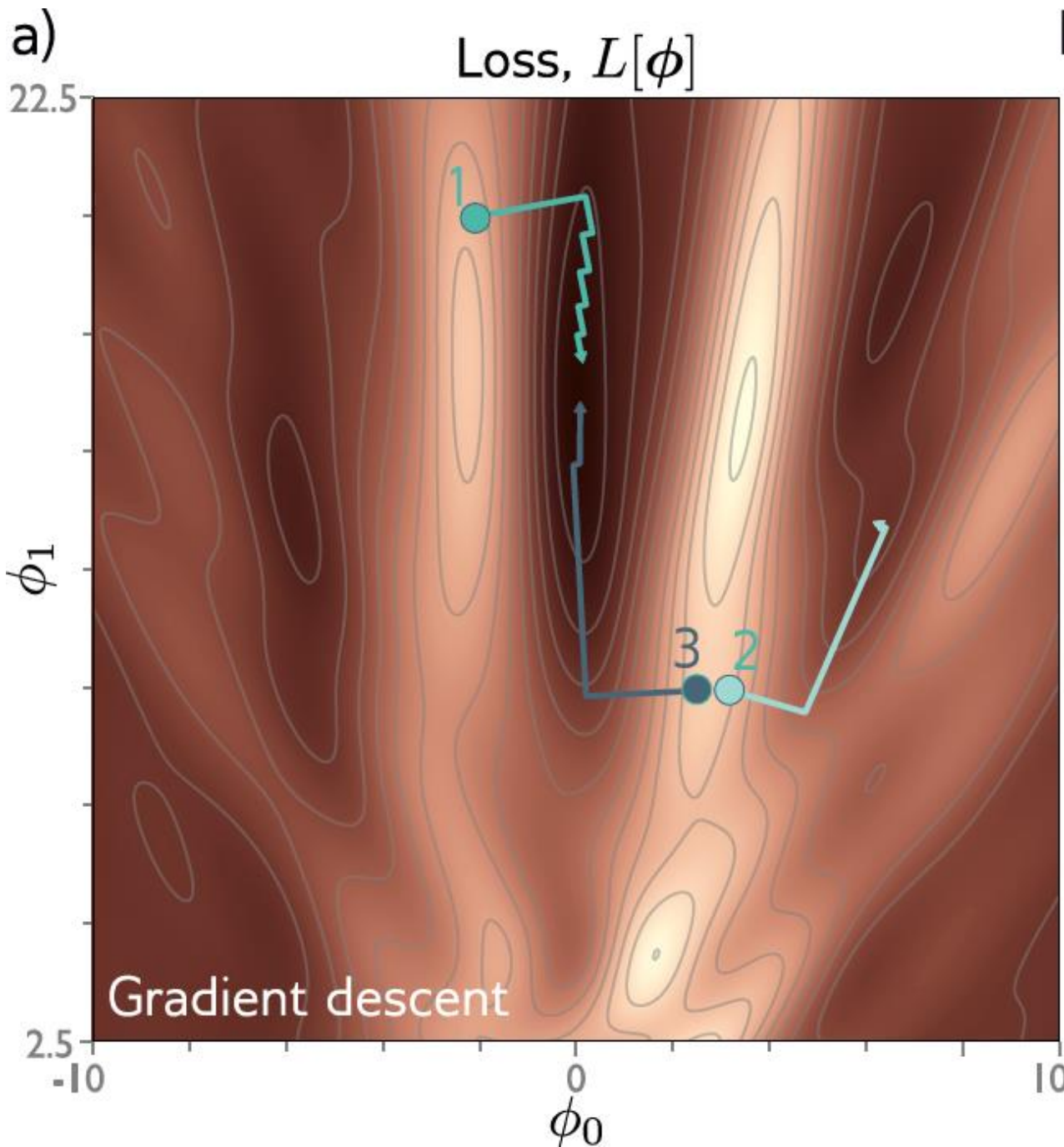
(Ex. 30 sample dataset, batch size 5)

Data Indices → [ 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29]  
Permute → [27 15 23 17 8 9 28 24 12 0 4 16 5 13 11 22 1 2 25 3 21 26 18 29 20 7 10 14 19 6]

30/5 = 6 batches  
per epoch

Epoch # 0-----  
Step 0, Batch # 0, Batch Range [0 1 2 3 4], Batch index: [27 15 23 17 8]  
Step 1, Batch # 1, Batch Range [5 6 7 8 9], Batch index: [ 9 28 24 12 0]  
Step 2, Batch # 2, Batch Range [10 11 12 13 14], Batch index: [ 4 16 5 13 11]  
Step 3, Batch # 3, Batch Range [15 16 17 18 19], Batch index: [22 1 2 25 3]  
Step 4, Batch # 4, Batch Range [20 21 22 23 24], Batch index: [21 26 18 29 20]  
Step 5, Batch # 5, Batch Range [25 26 27 28 29], Batch index: [ 7 10 14 19 6]  
Epoch # 1-----  
Step 6, Batch # 0, Batch Range [0 1 2 3 4], Batch index: [27 15 23 17 8]  
Step 7, Batch # 1, Batch Range [5 6 7 8 9], Batch index: [ 9 28 24 12 0]  
Step 8, Batch # 2, Batch Range [10 11 12 13 14], Batch index: [ 4 16 5 13 11]  
Step 9, Batch # 3, Batch Range [15 16 17 18 19], Batch index: [22 1 2 25 3]  
...

Batch Size 5



Stochastic gradient descent

Before (full batch descent)

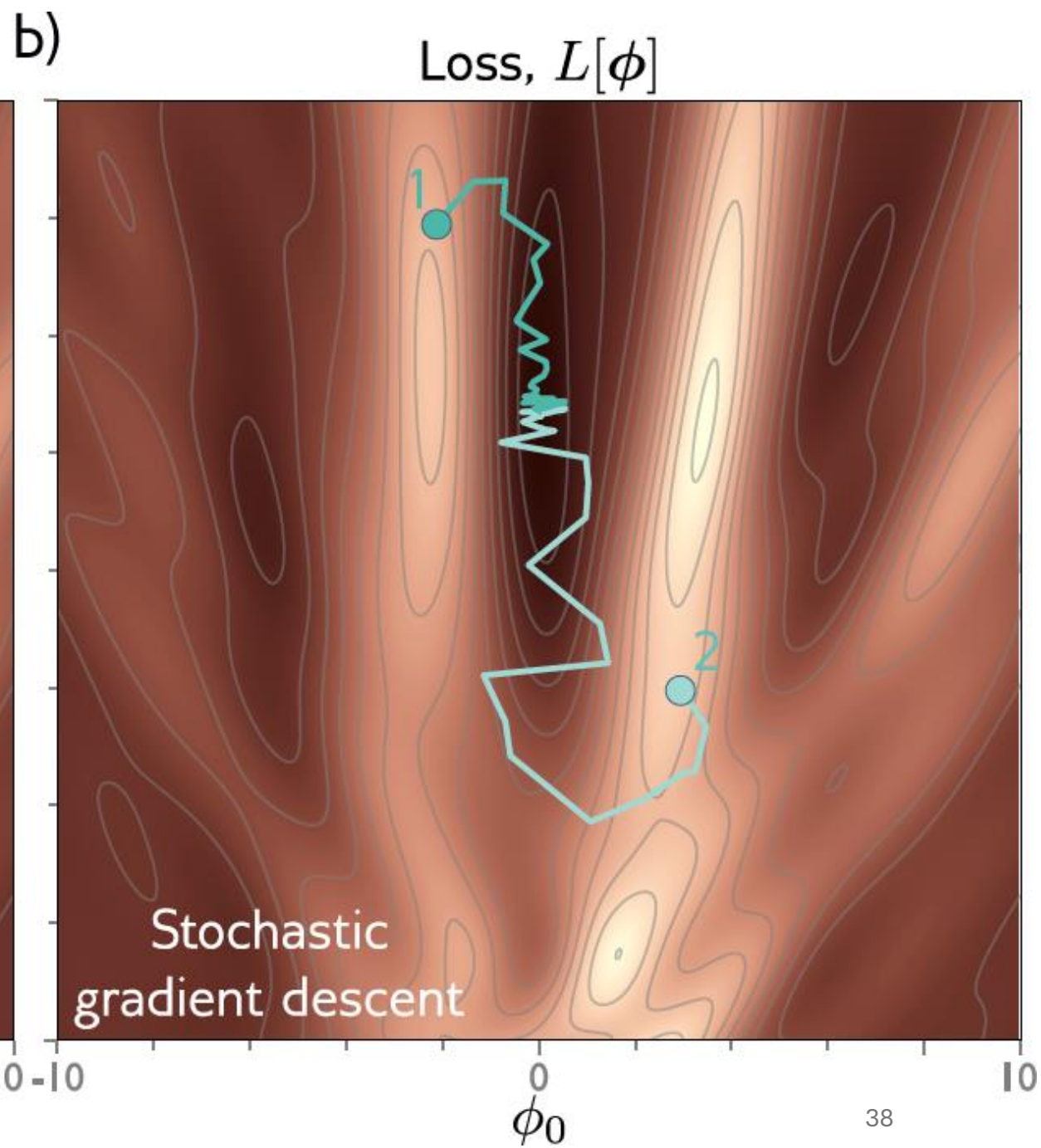
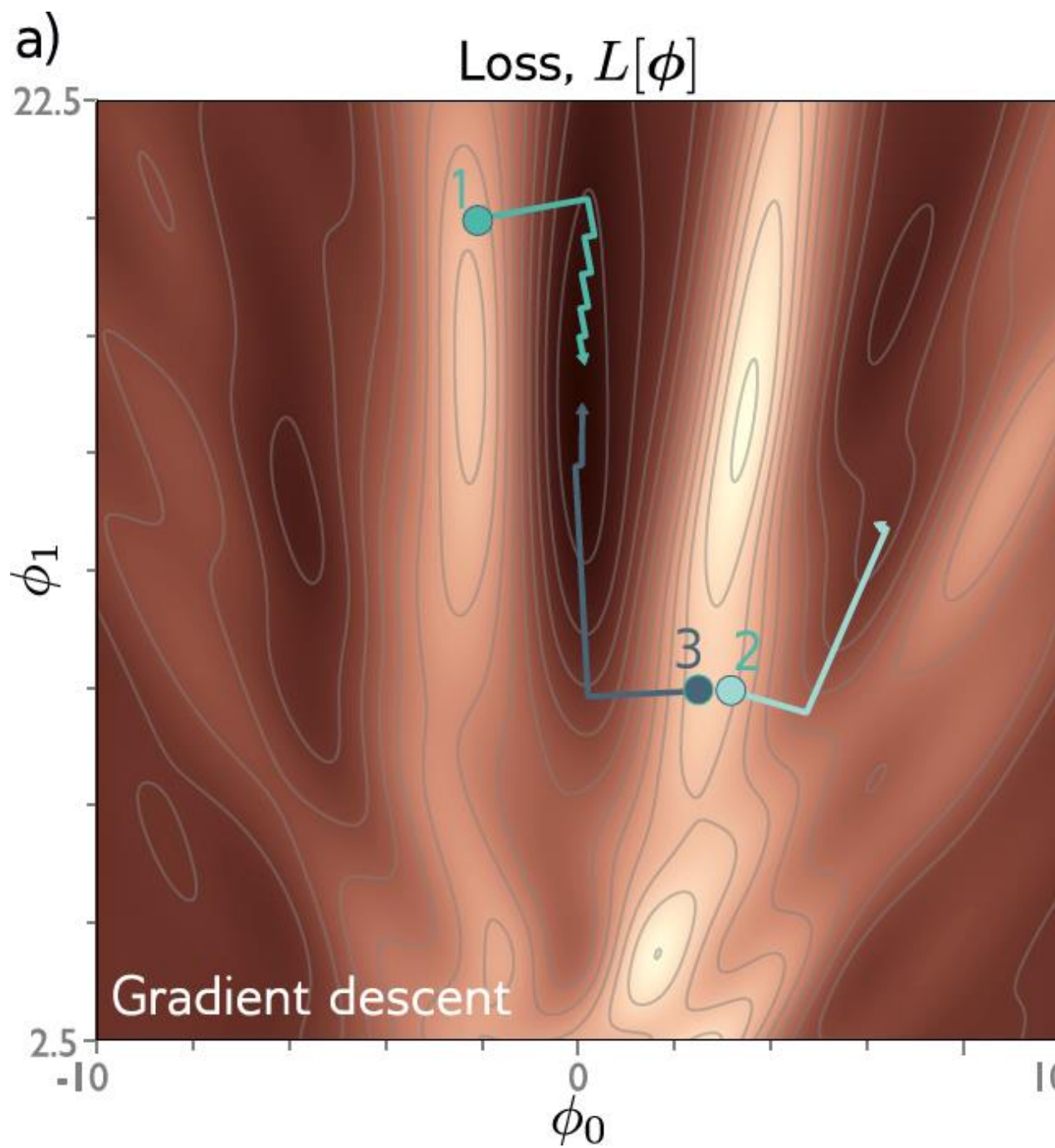
$$\phi_{t+1} \leftarrow \phi_t - \alpha \sum_{i=1}^I \frac{\partial \ell_i[\phi_t]}{\partial \phi},$$

After (SGD)

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

Fixed learning rate  $\alpha$





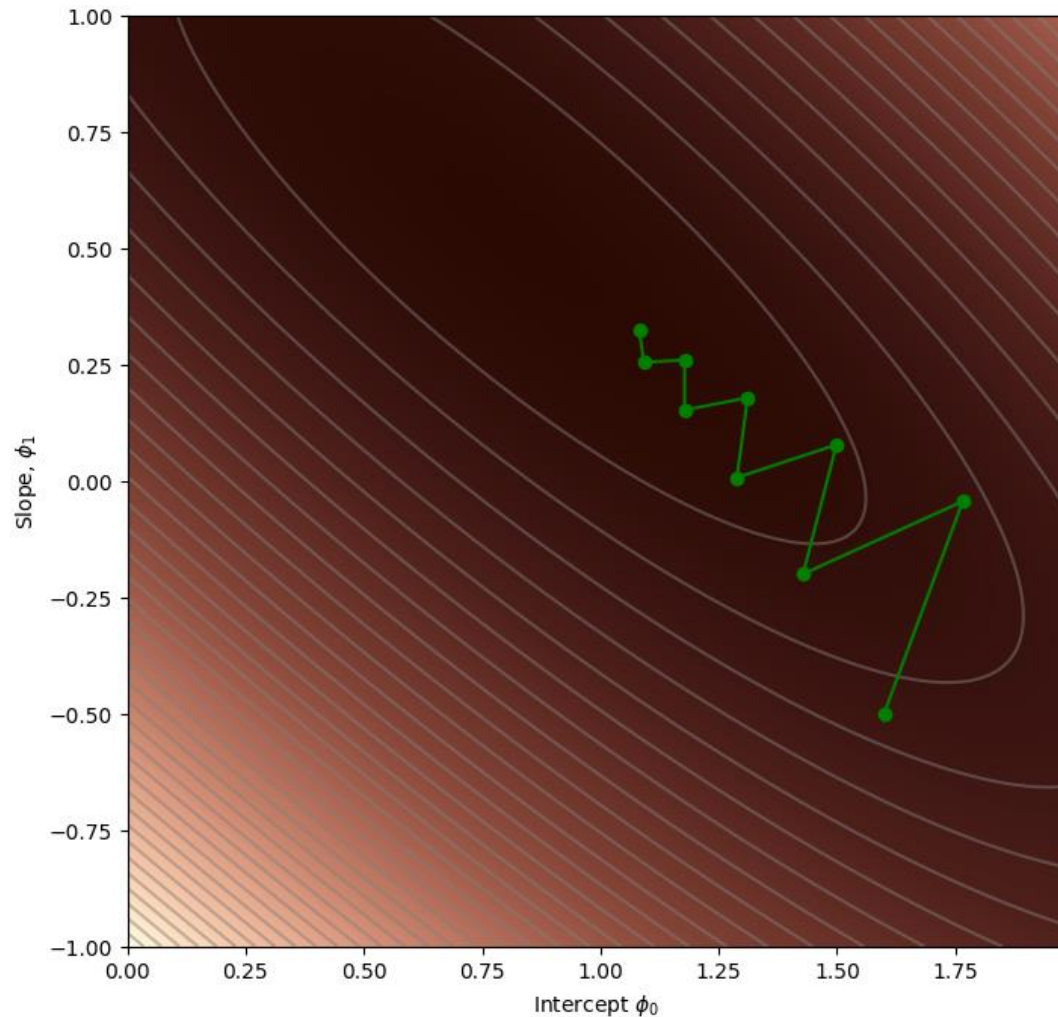
# Properties of SGD

- Can escape from local minima
- Adds noise, but still sensible updates as based on part of data
- Still 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

# Plan for Today

- Homework 3 post-mortem
- Gradient descent review
- Stochastic gradient descent (more formally)
- Momentum
- Adam

# Simple Gradient Descent



Think of analogy of a ball rolling down a hill.

Would it follow path like on the left?

Why/Why not? What's missing?



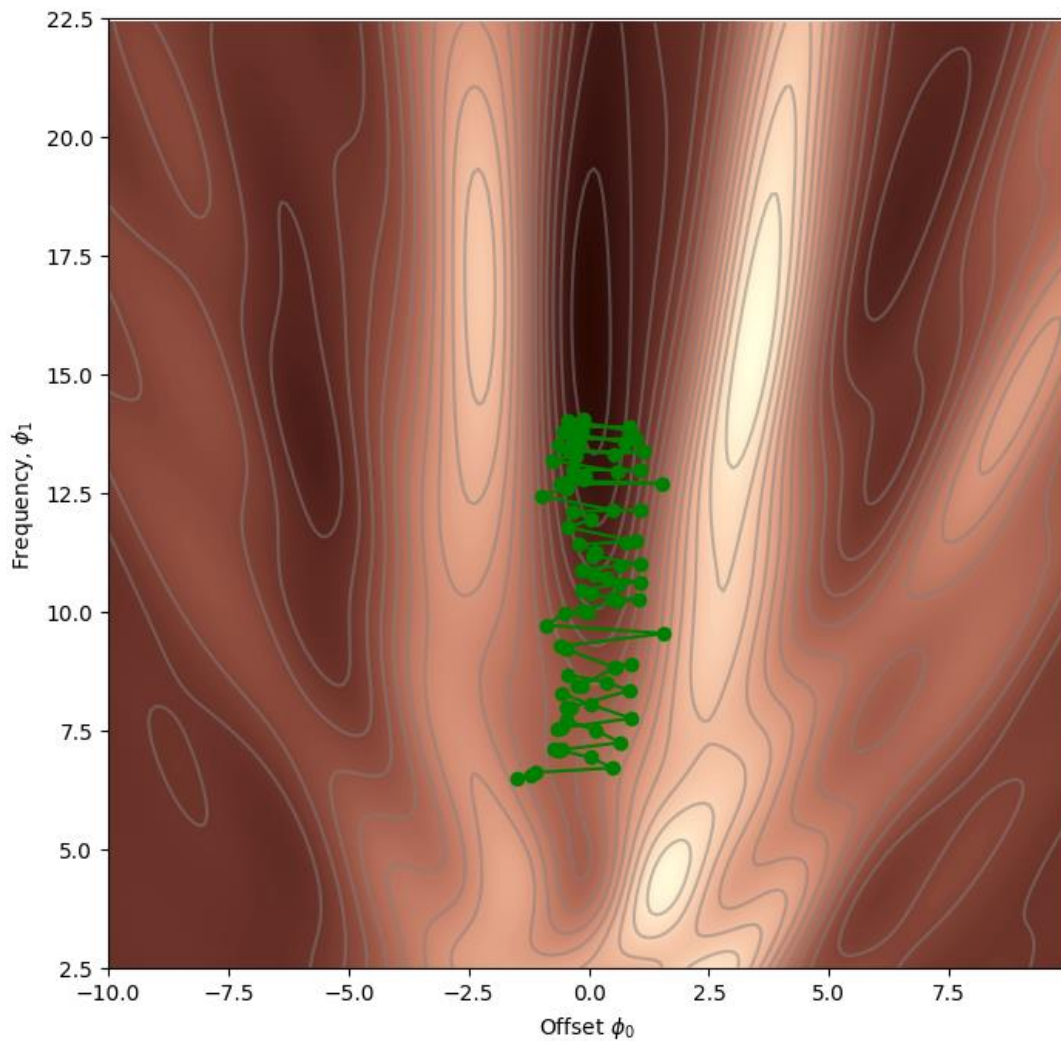
# Momentum

- Weighted sum of this gradient and previous gradient
- Not only influenced by gradient
- Changes more slowly over time

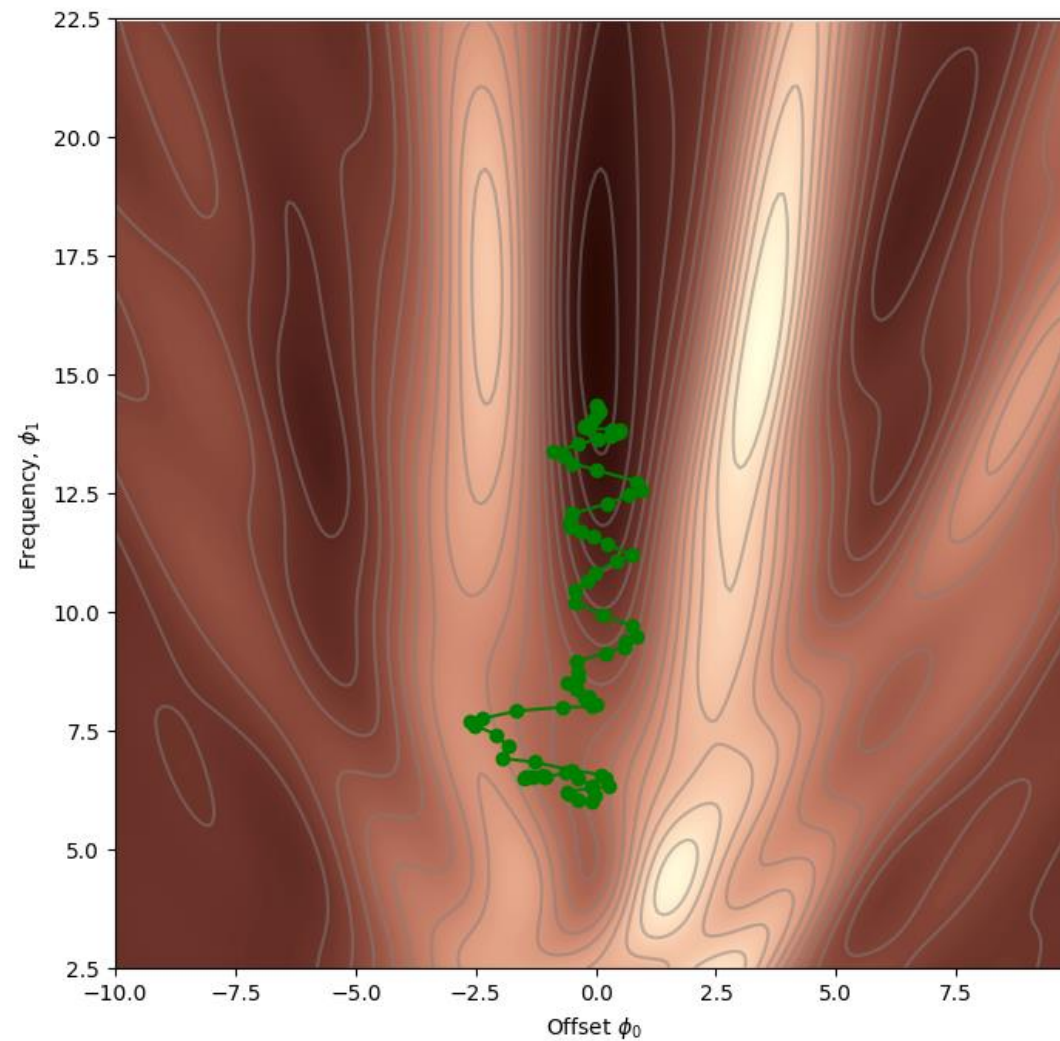
$$\mathbf{m}_{t+1} \leftarrow \beta \cdot \mathbf{m}_t + (1 - \beta) \sum_{i \in \mathcal{B}_t} \frac{\partial \ell_i[\phi_t]}{\partial \phi}$$
$$\phi_{t+1} \leftarrow \phi_t - \alpha \cdot \mathbf{m}_{t+1}$$

Still in batches.

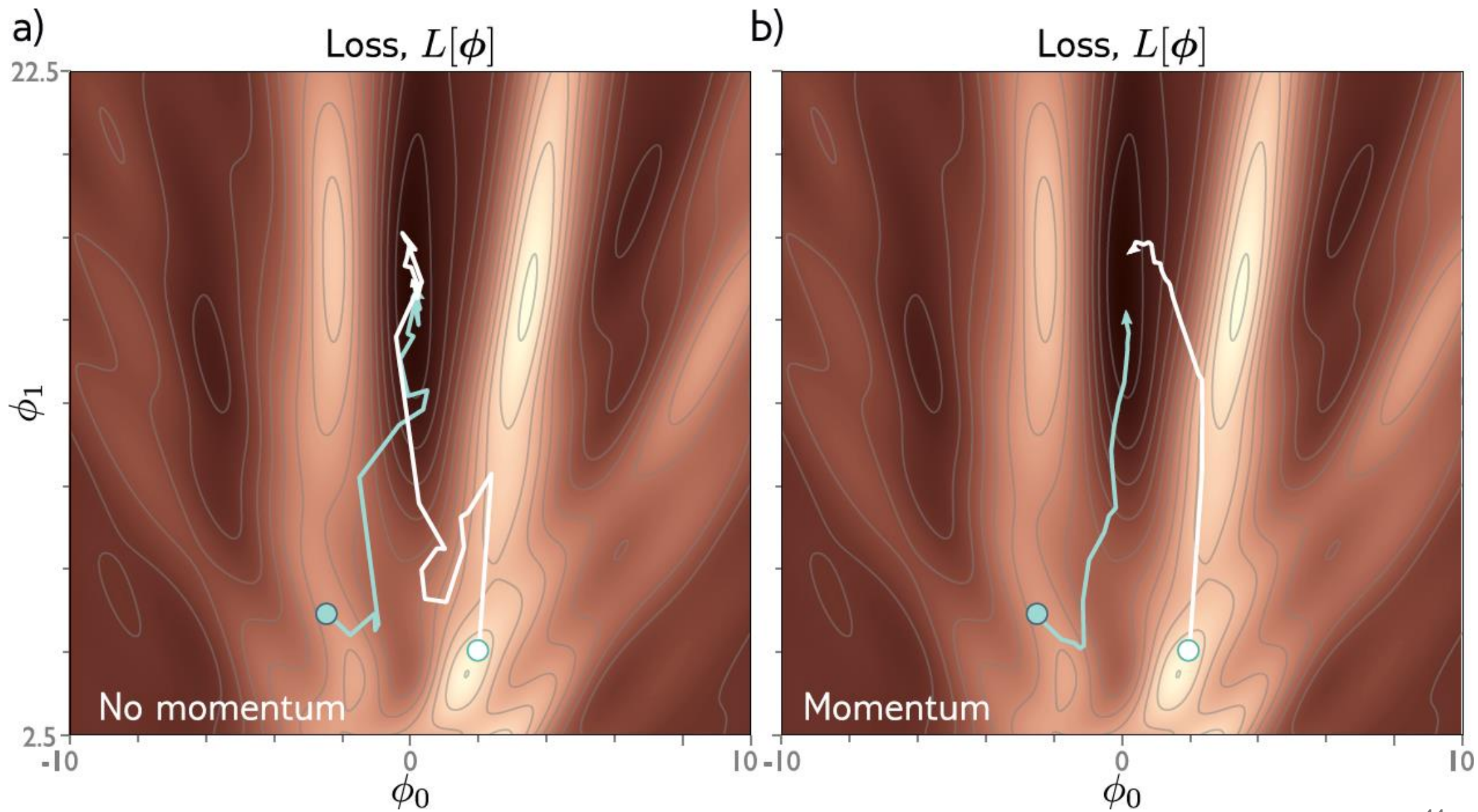
# Without and With Momentum



Without Momentum, Loss = 1.31



With Momentum, Loss = 0.96



# Nesterov accelerated momentum

- Momentum smooths out gradient of current location

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

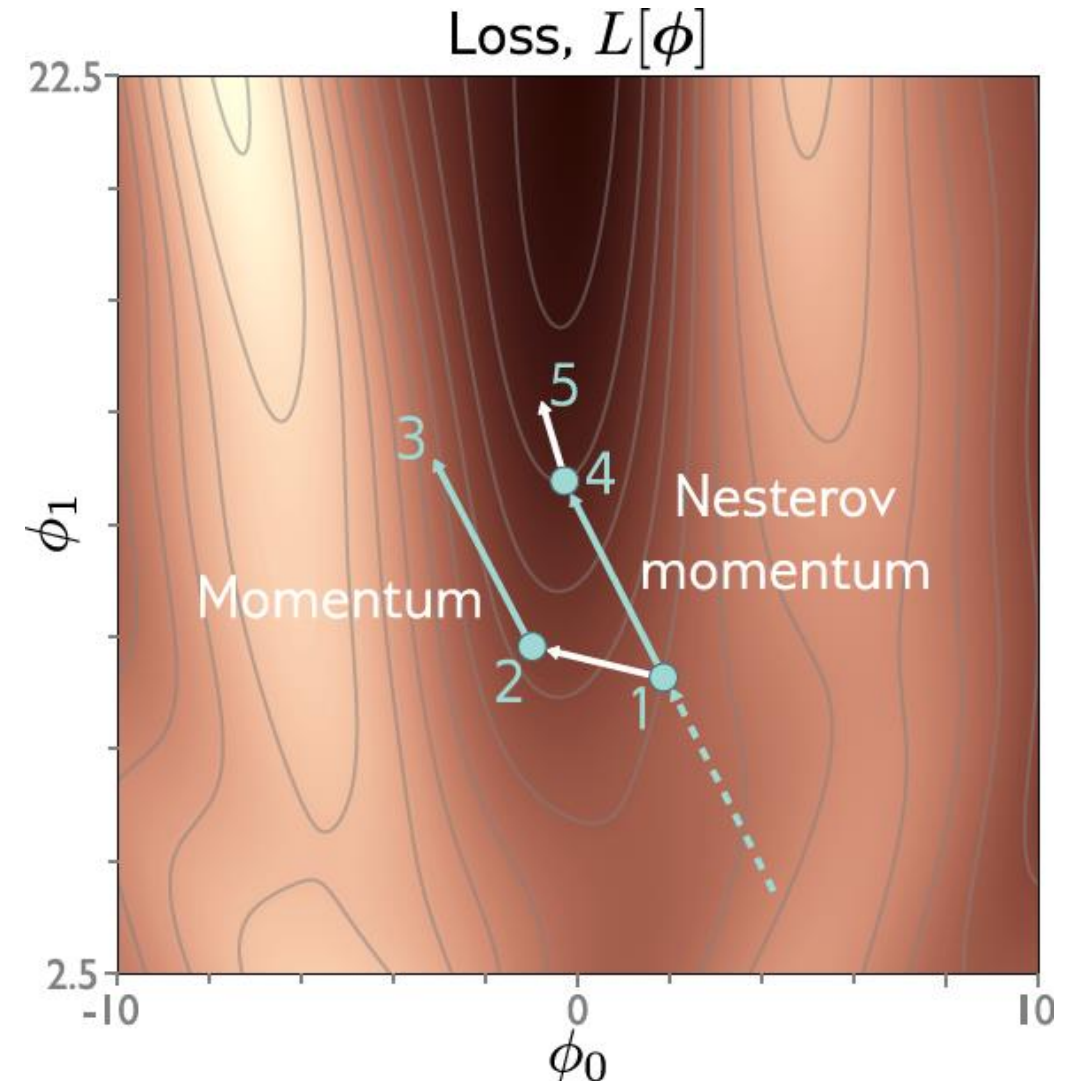
$$\phi_{t+1} \leftarrow \phi_t - \alpha \cdot \mathbf{m}_{t+1}$$

- Alternative, smooth out gradient of where we think we will be!

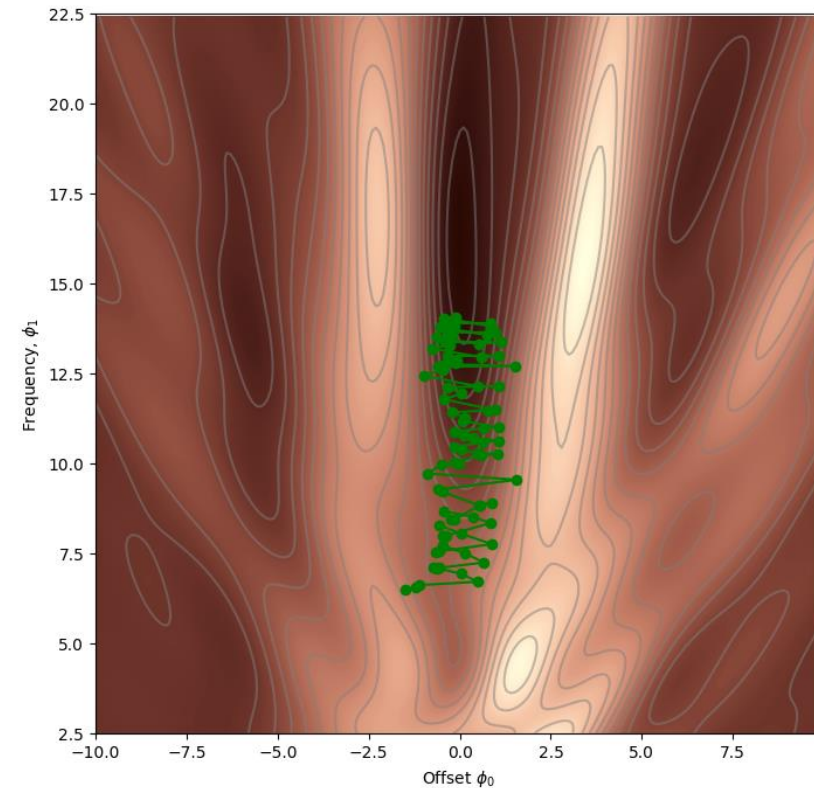
$$\mathbf{m}_{t+1} \leftarrow \beta \cdot \mathbf{m}_t + (1 - \beta) \sum_{i \in \mathcal{B}_t} \frac{\partial \ell_i[\phi_t - \alpha \cdot \mathbf{m}_t]}{\partial \phi}$$

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

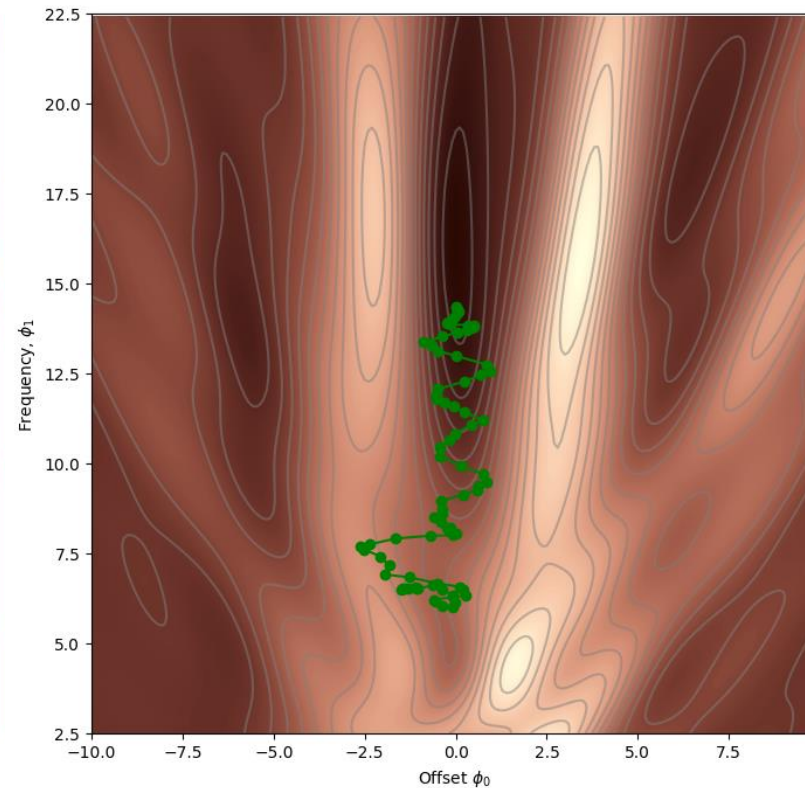
Still in batches.



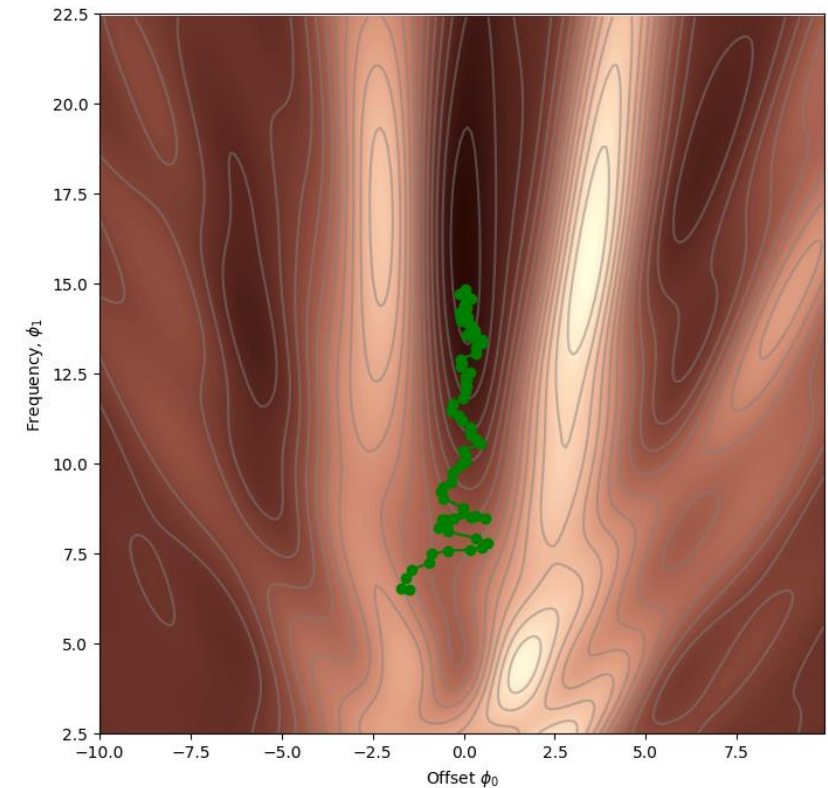
# Nesterov Momentum



Without Momentum, Loss =  
1.31



With Momentum, Loss =  
0.96



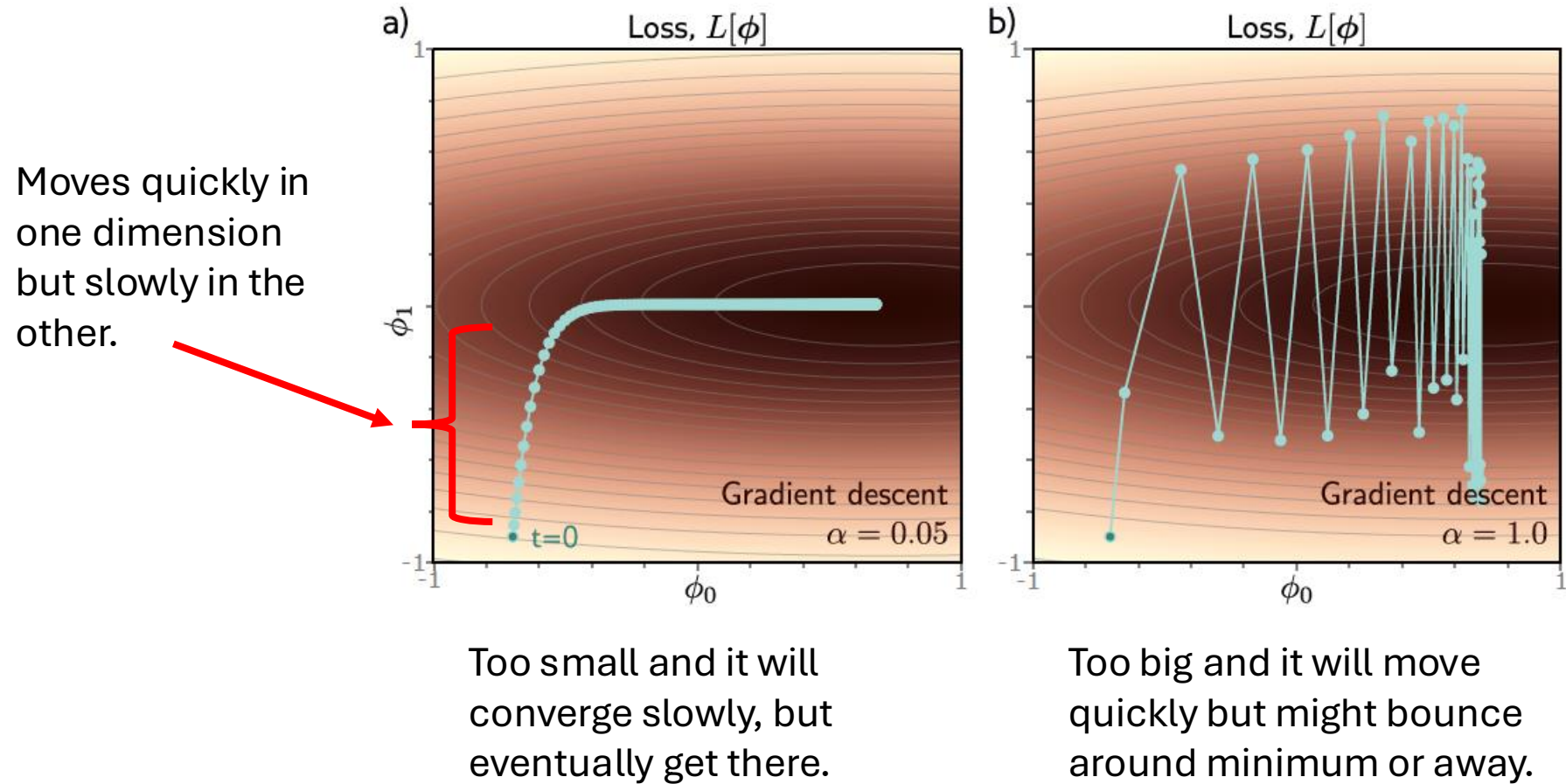
Nesterov Momentum, Loss =  
0.80

# Plan for Today

- Homework 3 Post-Mortem
- Gradient Descent Review
- Stochastic gradient descent (more formally)
- Momentum
- Adam



# The challenge with fixed step sizes



# Solution Part 1: Unit Vector Gradients

- Measure gradient  $\mathbf{m}_{t+1}$  and squared magnitude of gradient  $\mathbf{v}_{t+1}$

$$m_{t+1} \leftarrow \frac{\partial L[\phi_t]}{\partial \phi}$$
$$v_{t+1} \leftarrow \left| \frac{\partial L[\phi_t]}{\partial \phi} \right|^2$$

- Normalize:

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

$\alpha$  is the learning rate

$\epsilon$  is a small constant to prevent div by 0

Square, sqrt and div are all pointwise



# Solution Part 1: Unit Vector gradients

- Measure gradient  $\mathbf{m}_{t+1}$  and squared magnitude of gradient  $\mathbf{v}_{t+1}$

$$m_{t+1} \leftarrow \frac{\partial L[\phi_t]}{\partial \phi}$$
$$v_{t+1} \leftarrow \left| \frac{\partial L[\phi_t]}{\partial \phi} \right|^2$$

- Normalize:

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

$\alpha$  is the learning rate

$\epsilon$  is a small constant to prevent div by 0

Square, sqrt and div are all pointwise

Dividing by the magnitude, so normalized to unit vector.

# Solution Part 1: Unit Vector gradients

- Measure mean and pointwise squared gradient

$$m_{t+1} \leftarrow \frac{\partial L[\phi_t]}{\partial \phi}$$
$$v_{t+1} \leftarrow \left| \frac{\partial L[\phi_t]}{\partial \phi} \right|^2$$

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

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

$$v_{t+1} = 3^2 + (-2)^2 + 5^2 = 38$$

- Normalize:

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

$$\frac{m_{t+1}}{\sqrt{v_{t+1}} + \epsilon} \approx \begin{bmatrix} +0.49 \\ -0.32 \\ +0.81 \end{bmatrix}$$

Large gradient components suppress other gradient components!

# Solution Part 2: Normalized gradients

- Measure gradient  $\mathbf{m}_{t+1}$  and pointwise squared gradient  $\mathbf{v}_{t+1}$

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

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

- Normalize:

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

$\alpha$  is the learning rate

$\epsilon$  is a small constant to prevent div by 0

Square, sqrt and div are all pointwise

# Solution Part 2: Normalized gradients

- Measure gradient  $\mathbf{m}_{t+1}$  and pointwise squared gradient  $\mathbf{v}_{t+1}$

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

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

- Normalize:

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

$\alpha$  is the learning rate

$\epsilon$  is a small constant to prevent div by 0

Square, sqrt and div are all pointwise

Dividing by the positive root, so normalized to 1 and all that is left is the sign.

# Solution Part 2: Normalized gradients

- Measure mean and pointwise squared gradient

$$\mathbf{m}_{t+1} \leftarrow \frac{\partial L[\phi_t]}{\partial \phi} \qquad \mathbf{m}_{t+1} = \begin{bmatrix} 3.0 \\ -2.0 \\ 5.0 \end{bmatrix}$$

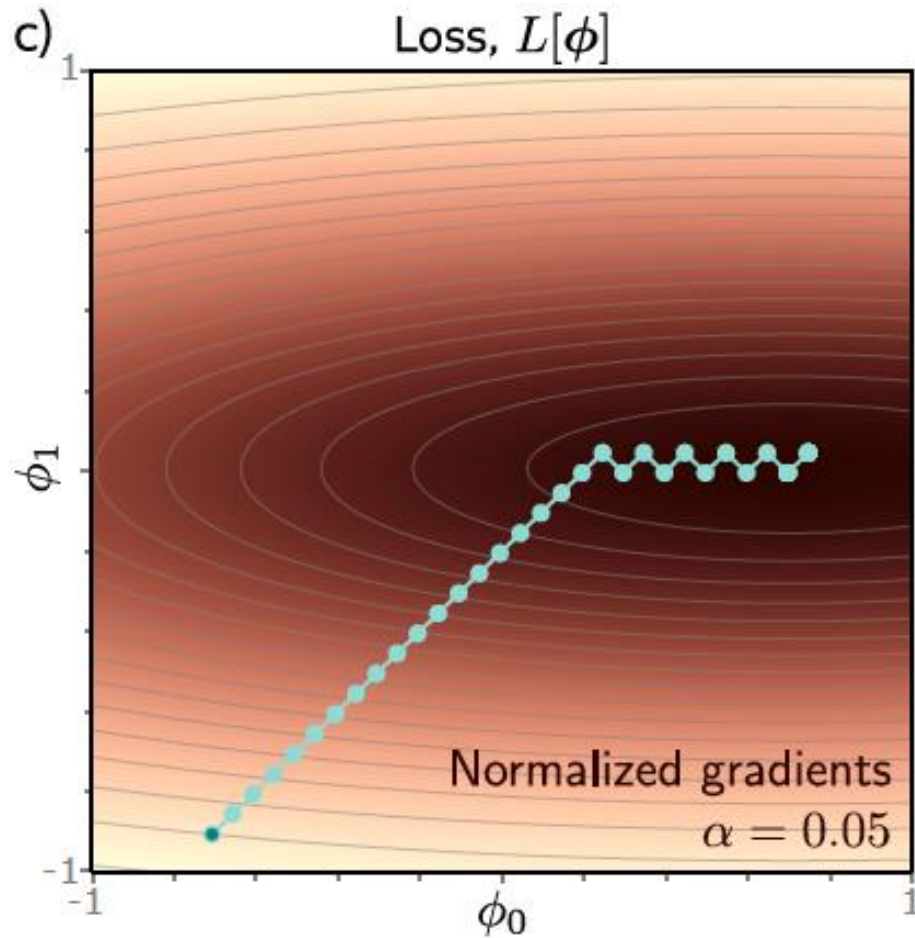
$$\mathbf{v}_{t+1} \leftarrow \frac{\partial L[\phi_t]^2}{\partial \phi} \qquad \mathbf{v}_{t+1} = \begin{bmatrix} 9.0 \\ 4.0 \\ 25.0 \end{bmatrix}$$

- Normalize:

$$\phi_{t+1} \leftarrow \phi_t - \alpha \cdot \frac{\mathbf{m}_{t+1}}{\sqrt{\mathbf{v}_{t+1}} + \epsilon} \qquad \frac{\mathbf{m}_{t+1}}{\sqrt{\mathbf{v}_{t+1}} + \epsilon} = \begin{bmatrix} 1.0 \\ -1.0 \\ 1.0 \end{bmatrix}$$



# Solution Part 2: Normalized gradients



- algorithm moves downhill a fixed distance  $\alpha$  along each coordinate
- makes good progress in both directions
- but will not converge unless it happens to land exactly at the minimum

# Adaptive moment estimation (Adam)

- Compute mean and pointwise squared gradients *with momentum*

$$\left\{ \begin{array}{l} \mathbf{m}_{t+1} \leftarrow \beta \cdot \mathbf{m}_t + (1 - \beta) \frac{\partial L[\phi_t]}{\partial \phi} \\ \mathbf{v}_{t+1} \leftarrow \gamma \cdot \mathbf{v}_t + (1 - \gamma) \left( \frac{\partial L[\phi_t]}{\partial \phi} \right)^2 \end{array} \right.$$

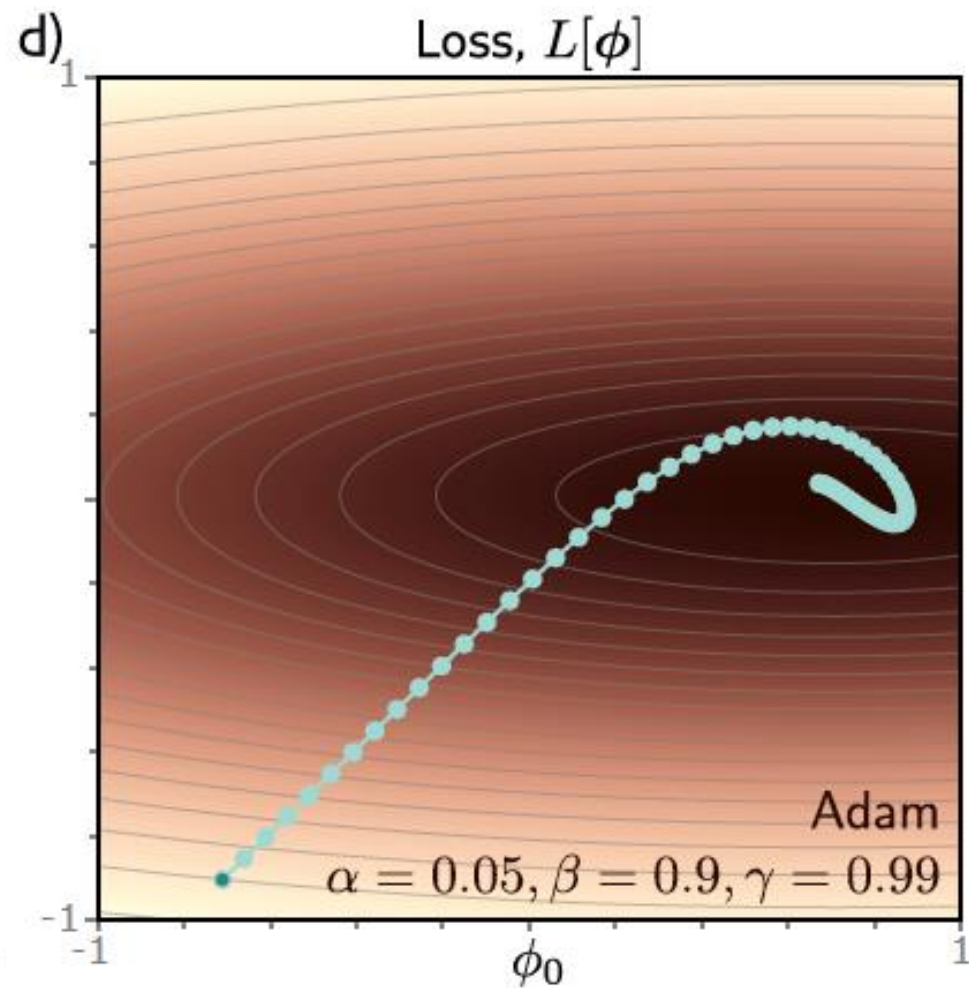
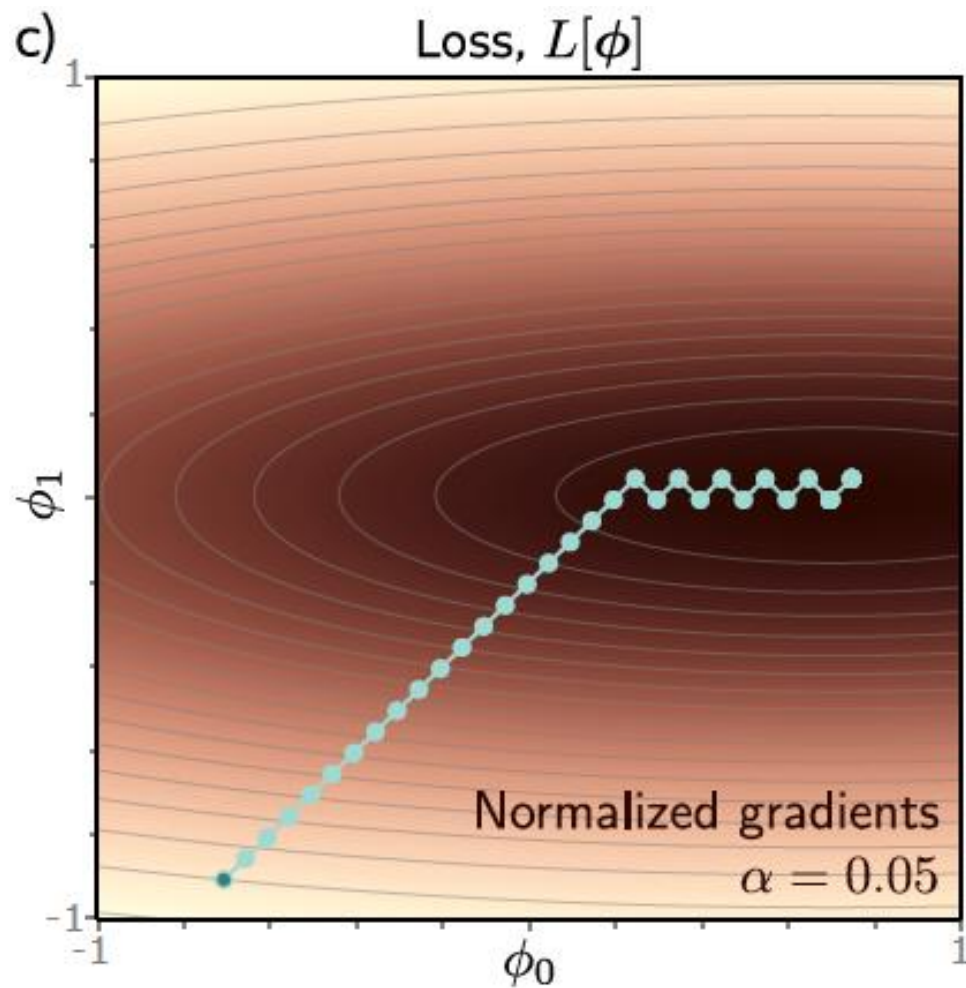
- Boost momentum near start of the sequence since they are initialized to zero

$$\left\{ \begin{array}{ll} \tilde{\mathbf{m}}_{t+1} \leftarrow \frac{\mathbf{m}_{t+1}}{1 - \beta^{t+1}} & \mathbf{m}_{t=0} = 0 \\ \tilde{\mathbf{v}}_{t+1} \leftarrow \frac{\mathbf{v}_{t+1}}{1 - \gamma^{t+1}} & \mathbf{v}_{t=0} = 0 \end{array} \right.$$

- Update the parameters

$$\left\{ \begin{array}{l} \phi_{t+1} \leftarrow \phi_t - \alpha \cdot \frac{\tilde{\mathbf{m}}_{t+1}}{\sqrt{\tilde{\mathbf{v}}_{t+1} + \epsilon}} \end{array} \right.$$

# Adaptive moment estimation (Adam)



# Other advantages of ADAM

- Gradients can diminish or grow deep into networks. ADAM balances out changes across depth of layers.
- Adam is less sensitive to the initial learning rate, so it doesn't need complex learning rate schedules.

# Additional Hyperparameters

- Choice of learning algorithm
  - SGD
  - Momentum
  - Nesterov Momentum
  - ADAM
- Learning rate
  - Fixed
  - Schedule
  - Loss dependent
- Momentum Parameters

# Recap

- **Gradient Descent** – Find a minimum for non-convex, complex loss functions
- **Stochastic Gradient Descent** – Save compute by calculating gradients in batches, which adds some noise to the search
- **(Nesterov) Momentum** – Add momentum to the gradient updates to smooth out abrupt gradient changes
- **ADAM** – Correct for imbalance between gradient components while providing some momentum