



# Fitting Models

DL4DS – Spring 2025

# Loss function

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

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

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

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

or for short:

$$L[\phi]$$

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

# Training

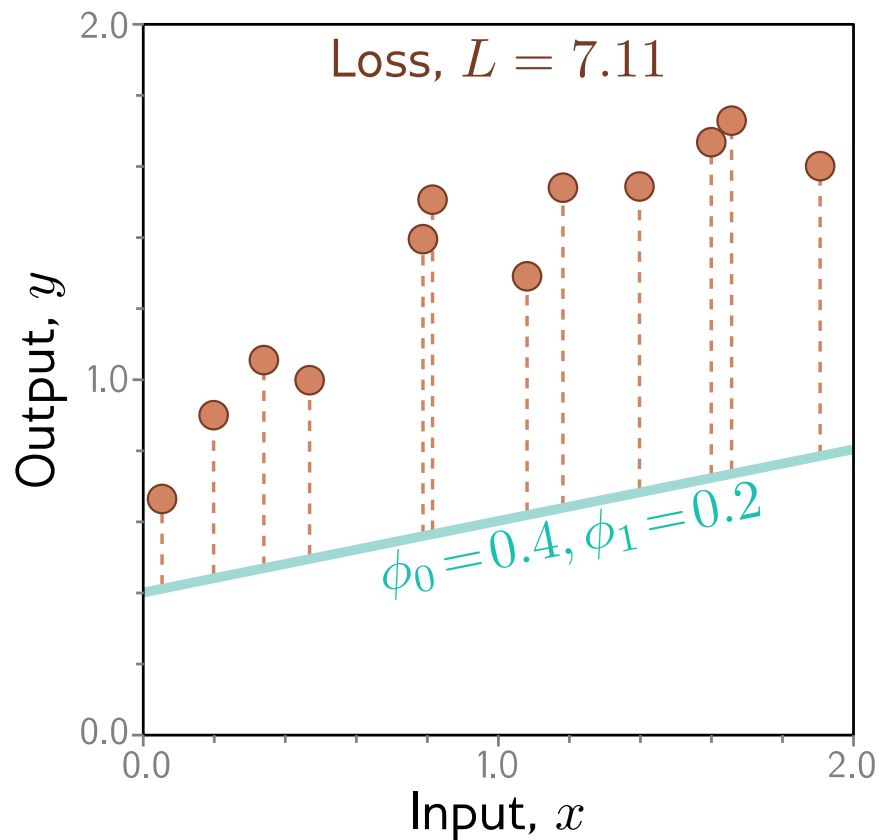
- Loss function:

$L[\phi]$  ← Returns a scalar that is smaller when model maps inputs to outputs better

- Find the parameters that minimize the loss:

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

# Example: 1D Linear regression loss function

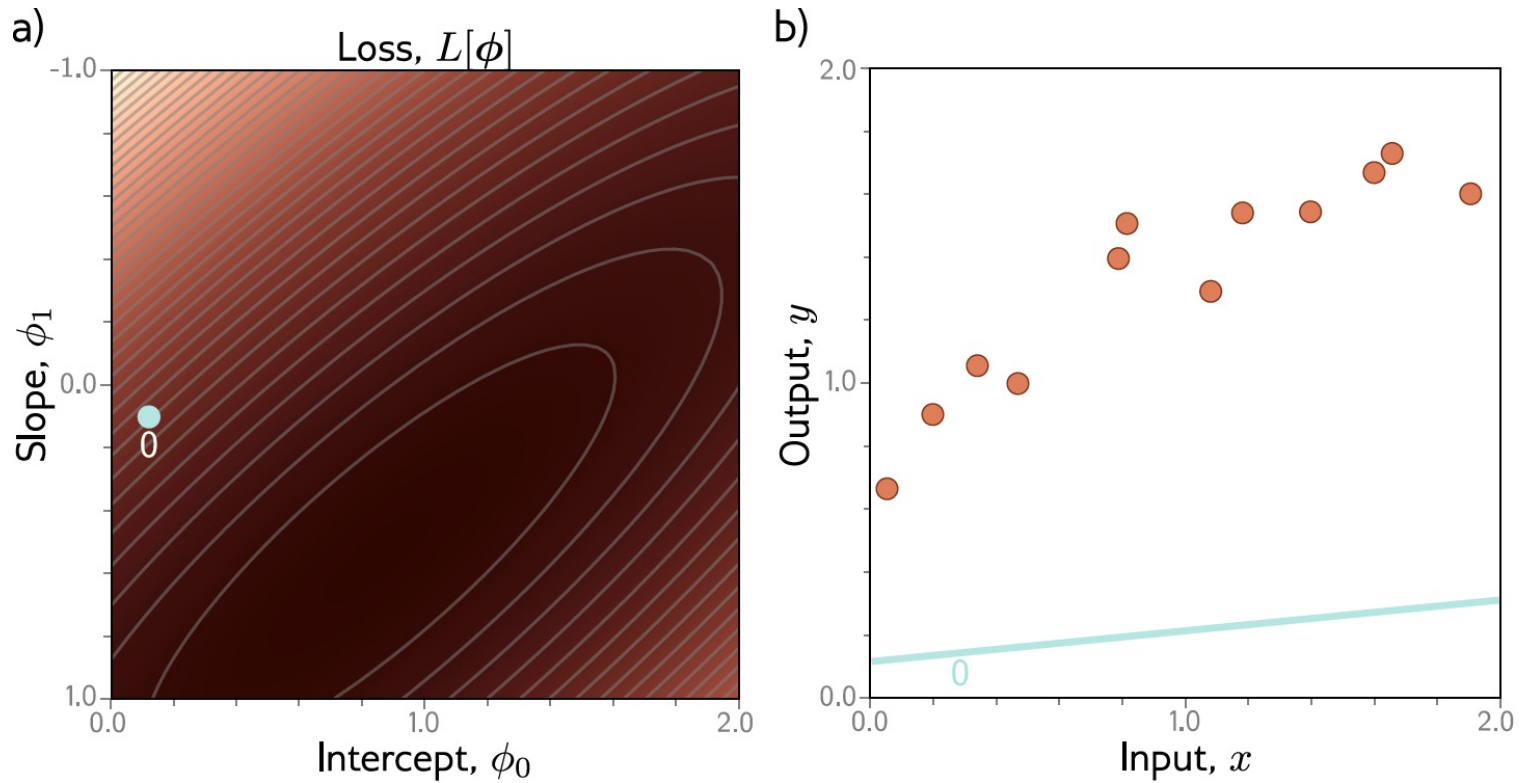


Loss function:

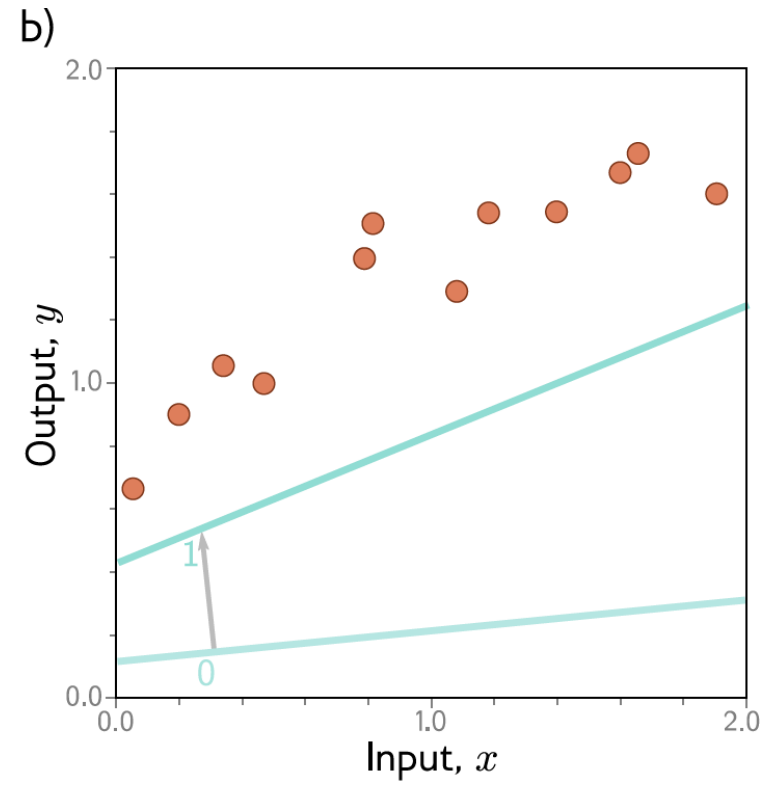
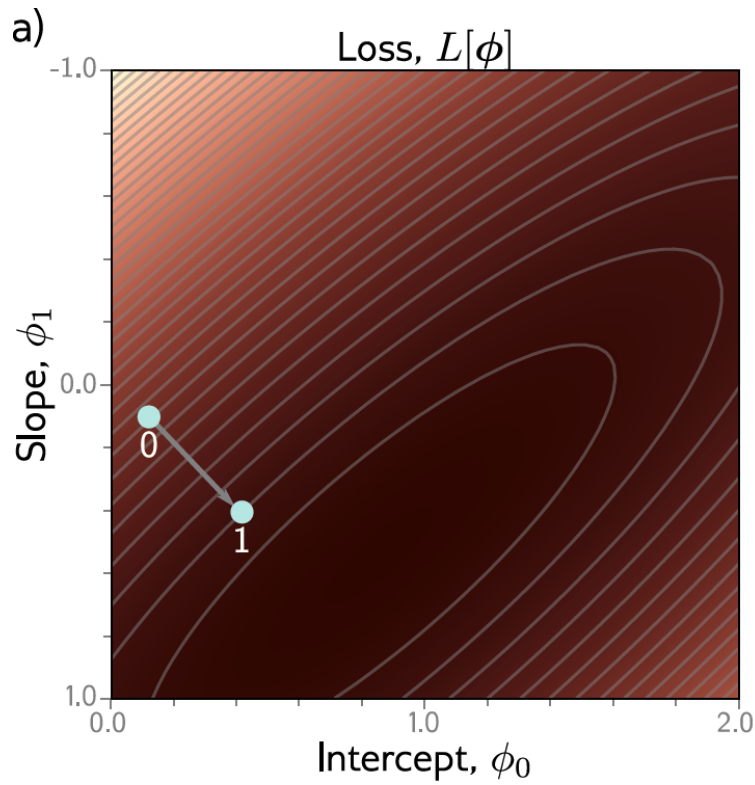
$$L[\phi] = \sum_{i=1}^I (f[x_i, \phi] - y_i)^2$$
$$= \sum_{i=1}^I (\phi_0 + \phi_1 x_i - y_i)^2$$

“Least squares loss function”

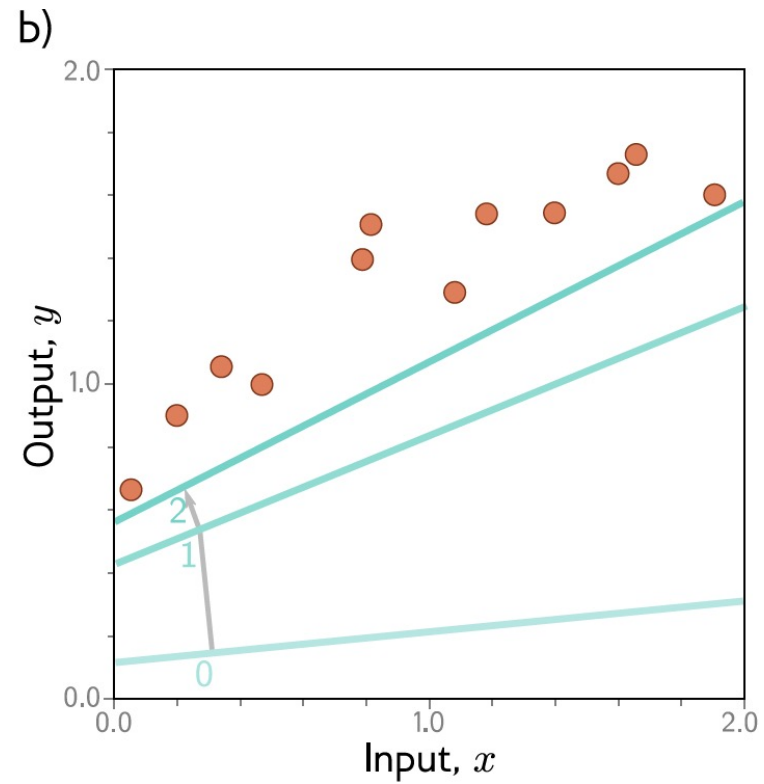
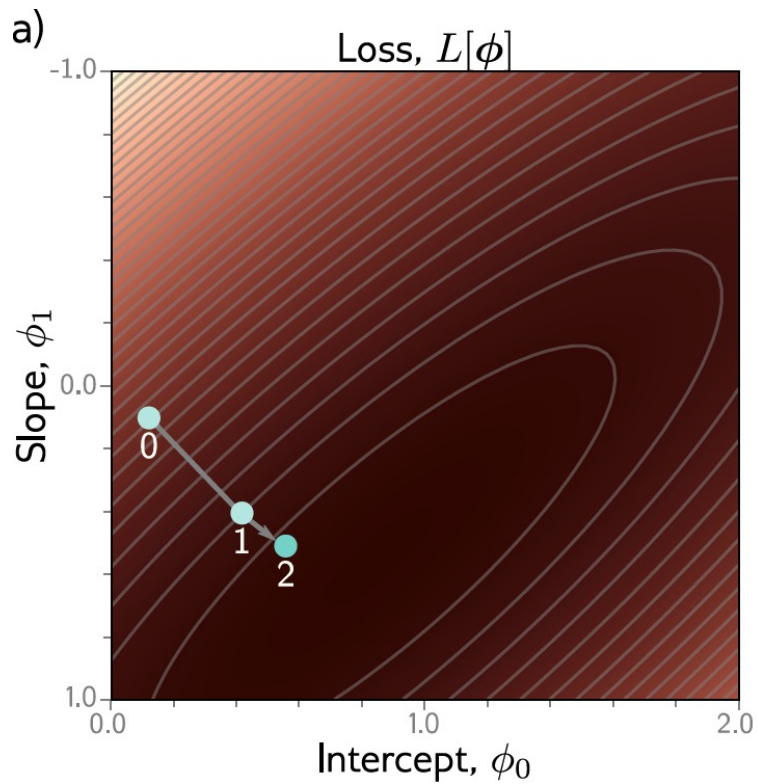
# Example: 1D Linear regression training



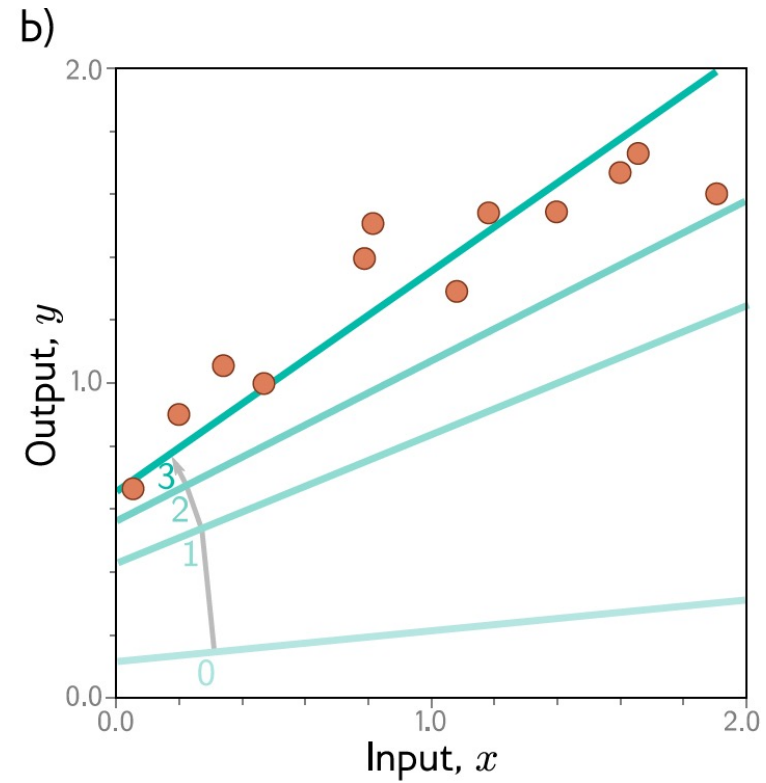
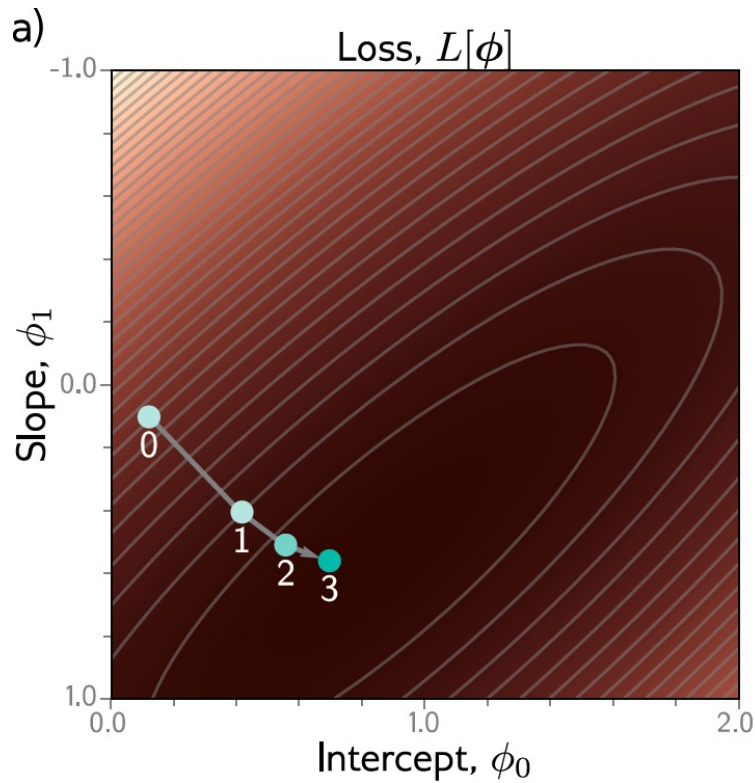
# Example: 1D Linear regression training



# Example: 1D Linear regression training

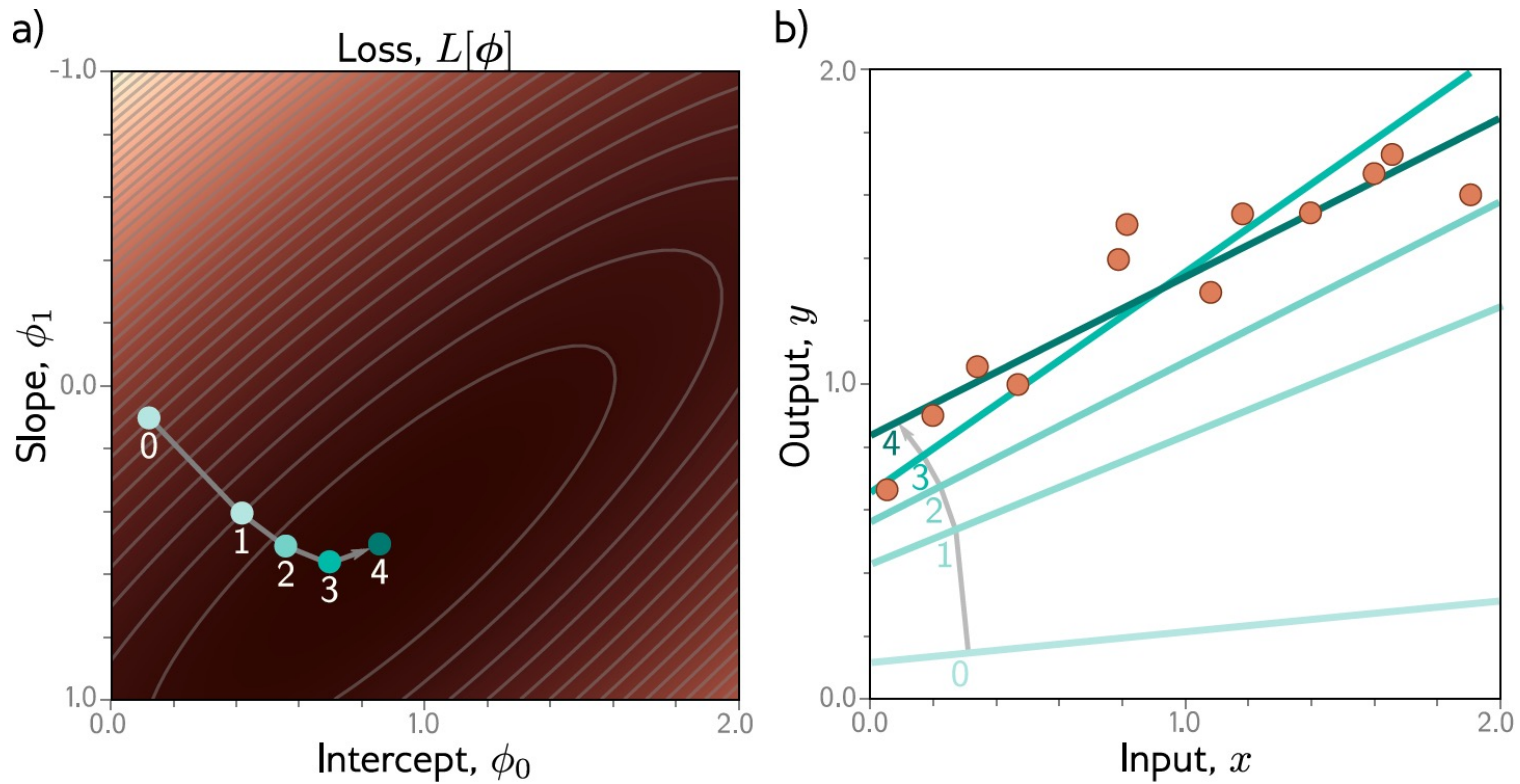


# Example: 1D Linear regression training





# Example: 1D Linear regression training



This technique is known as **gradient descent**

# Fitting models

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

# Definitions

- derivative

- quantifies the sensitivity of change of a function's output with respect to its input

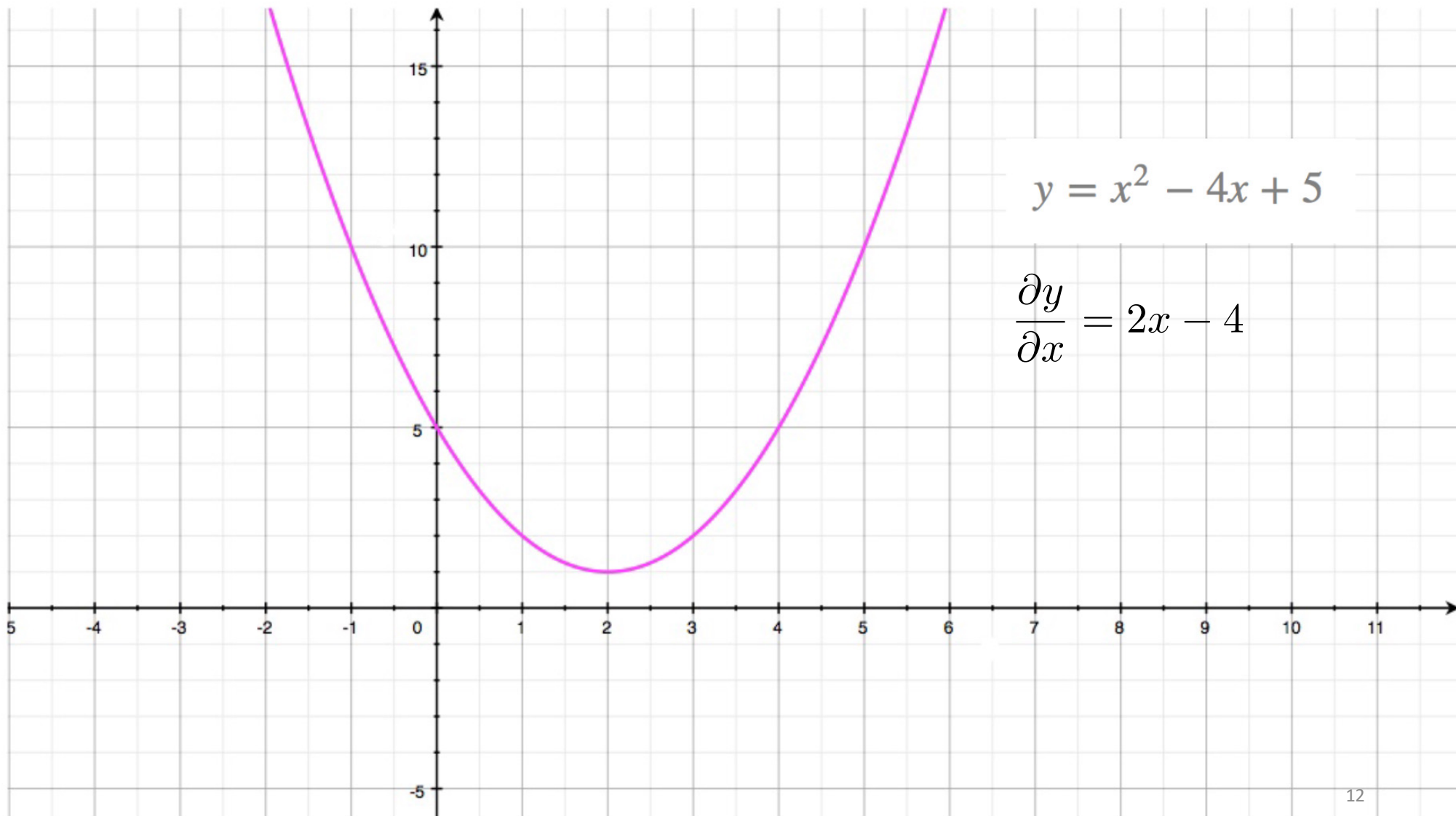
- a function is *differentiable* at a point  $a$ , if the limit

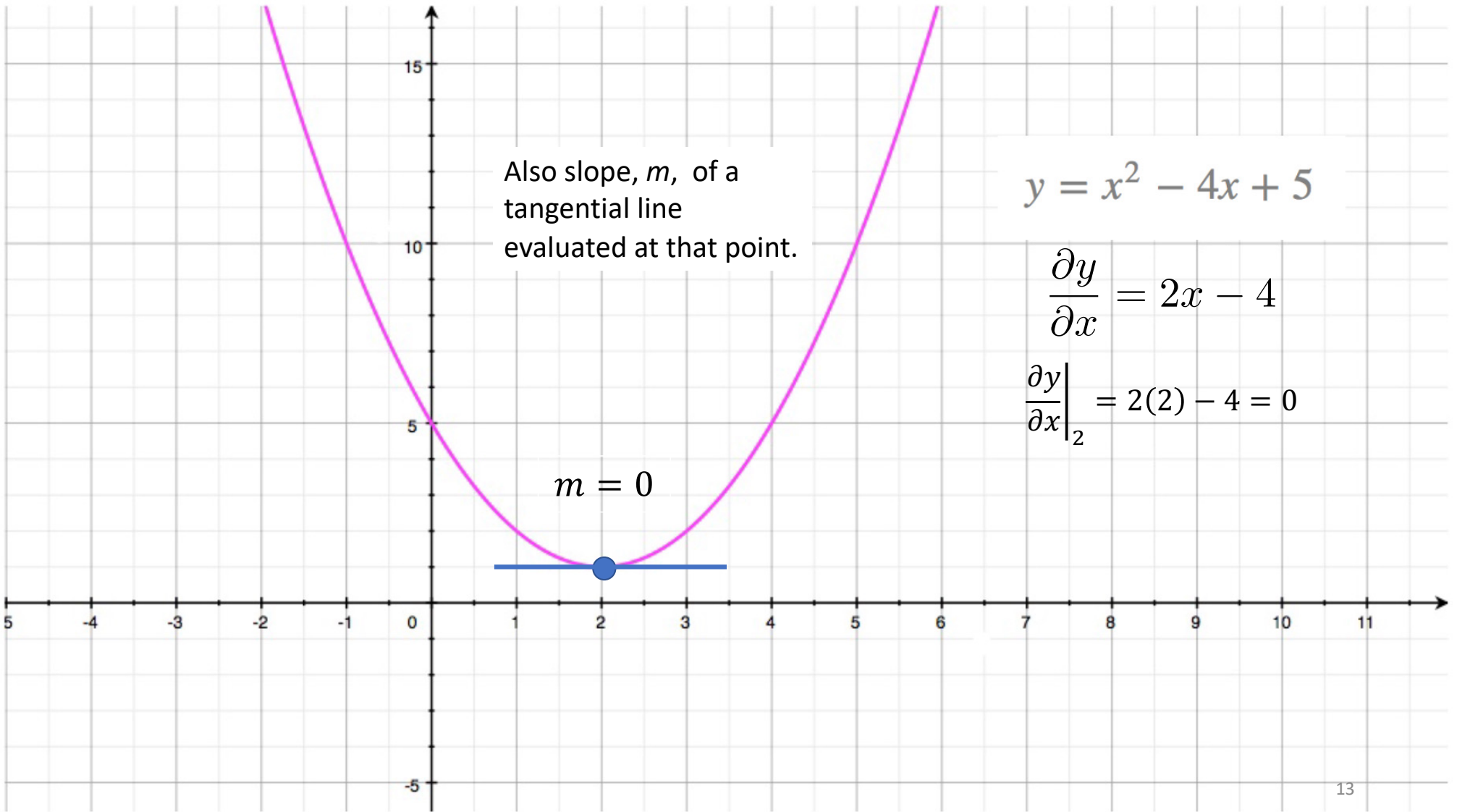
$$\lim_{h \rightarrow 0} \frac{f(a+h) - f(a)}{h} \text{ exists.}$$

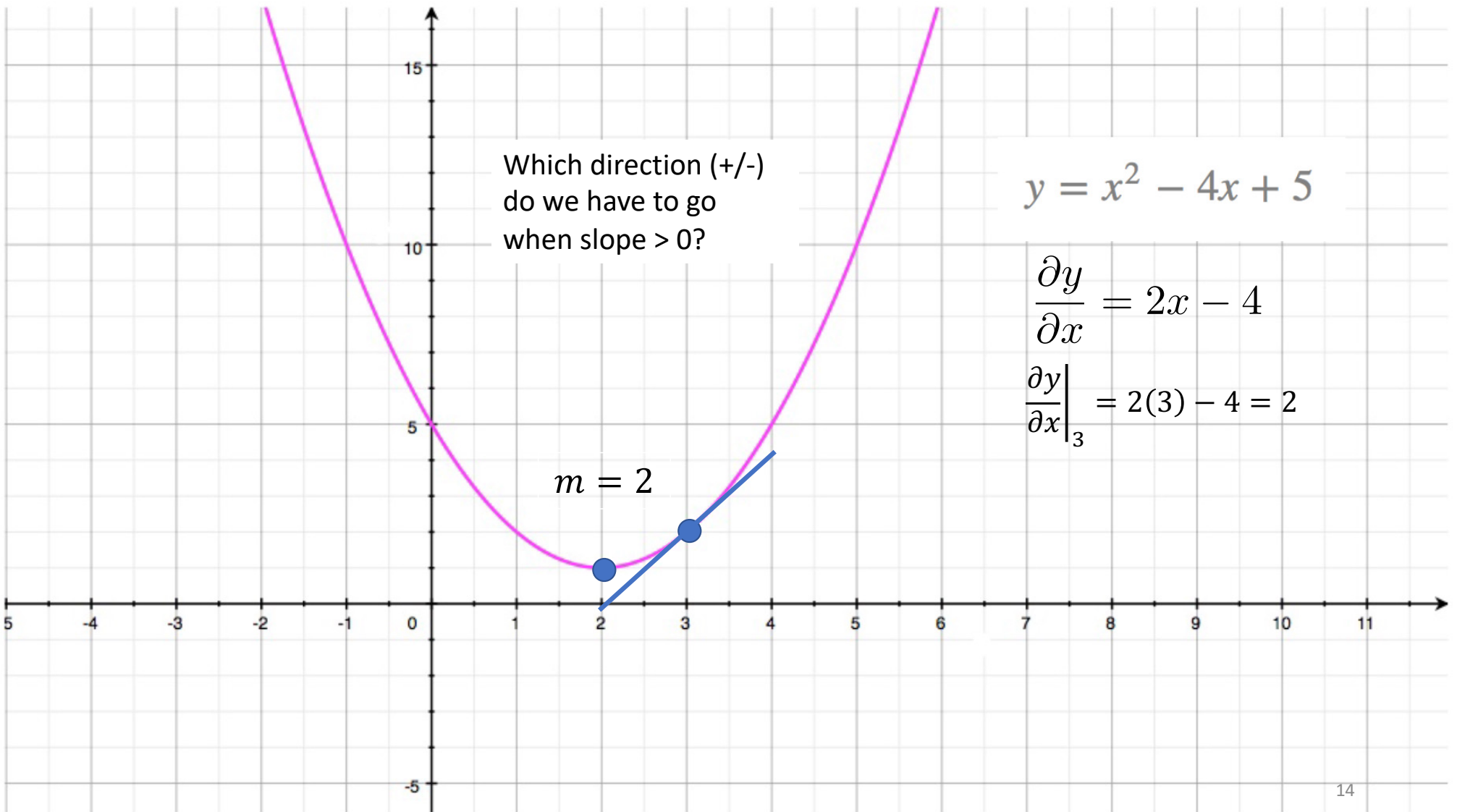
- You can approximate the derivative with this limit.

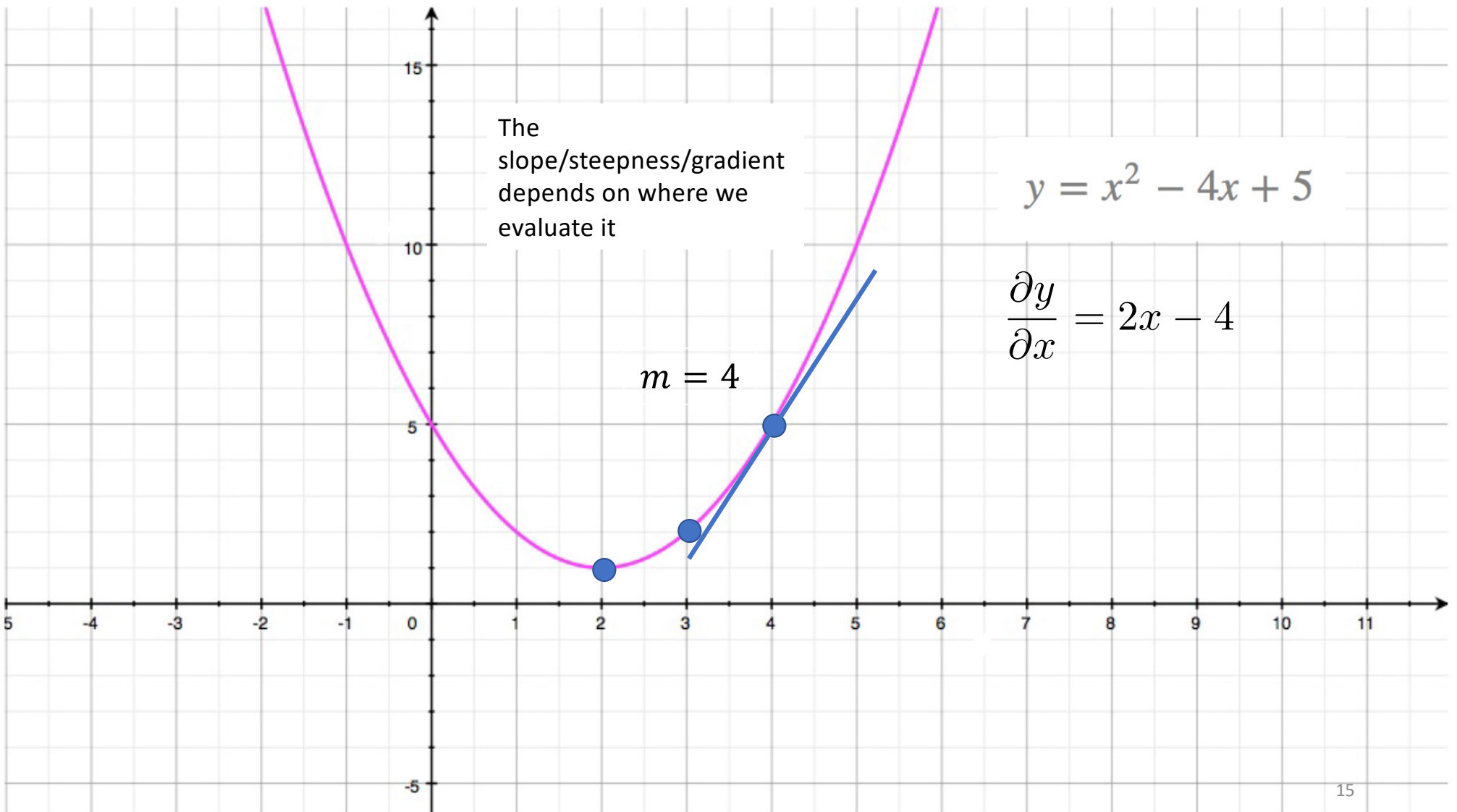
- gradient

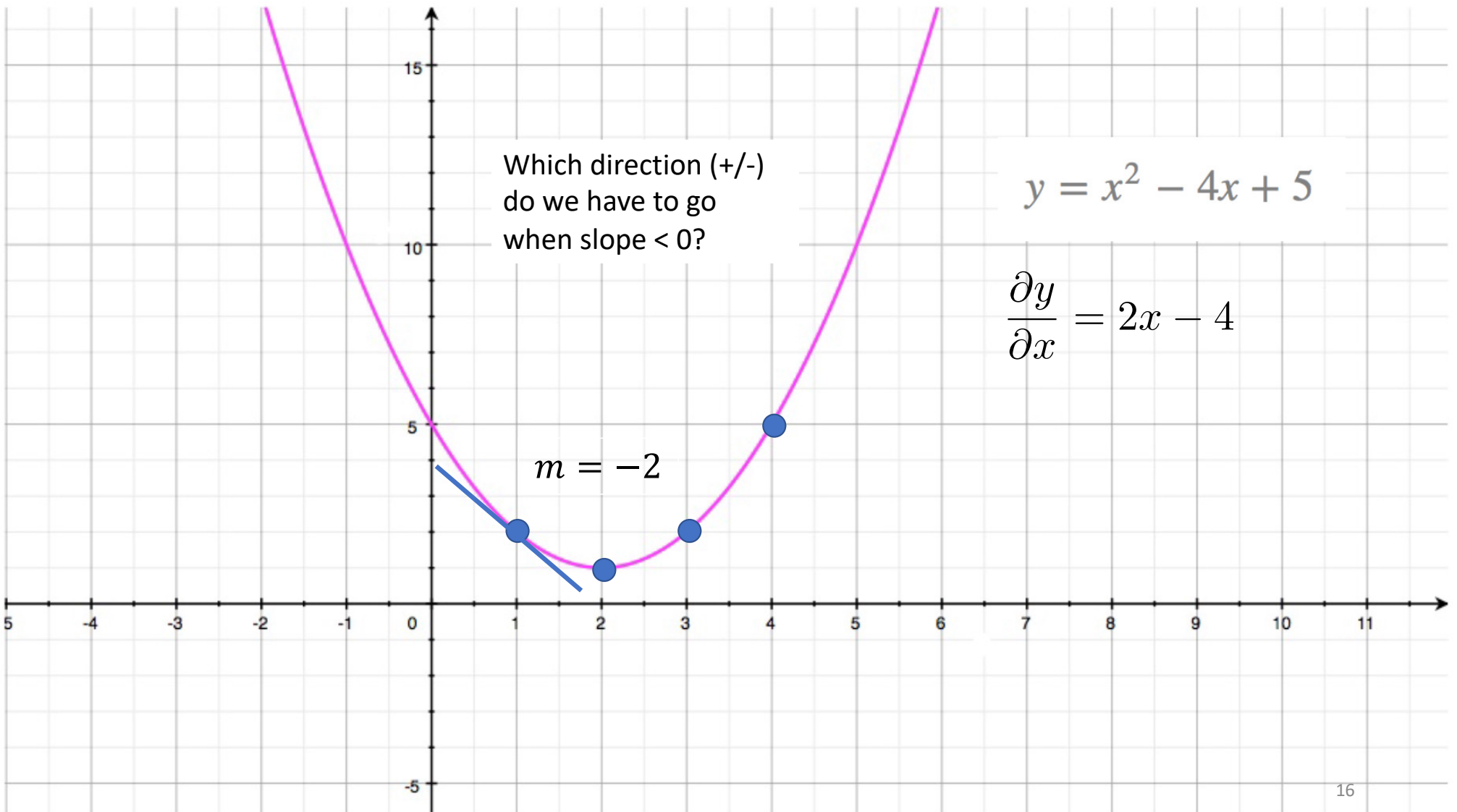
- the degree and direction of steepness of a graph at any point



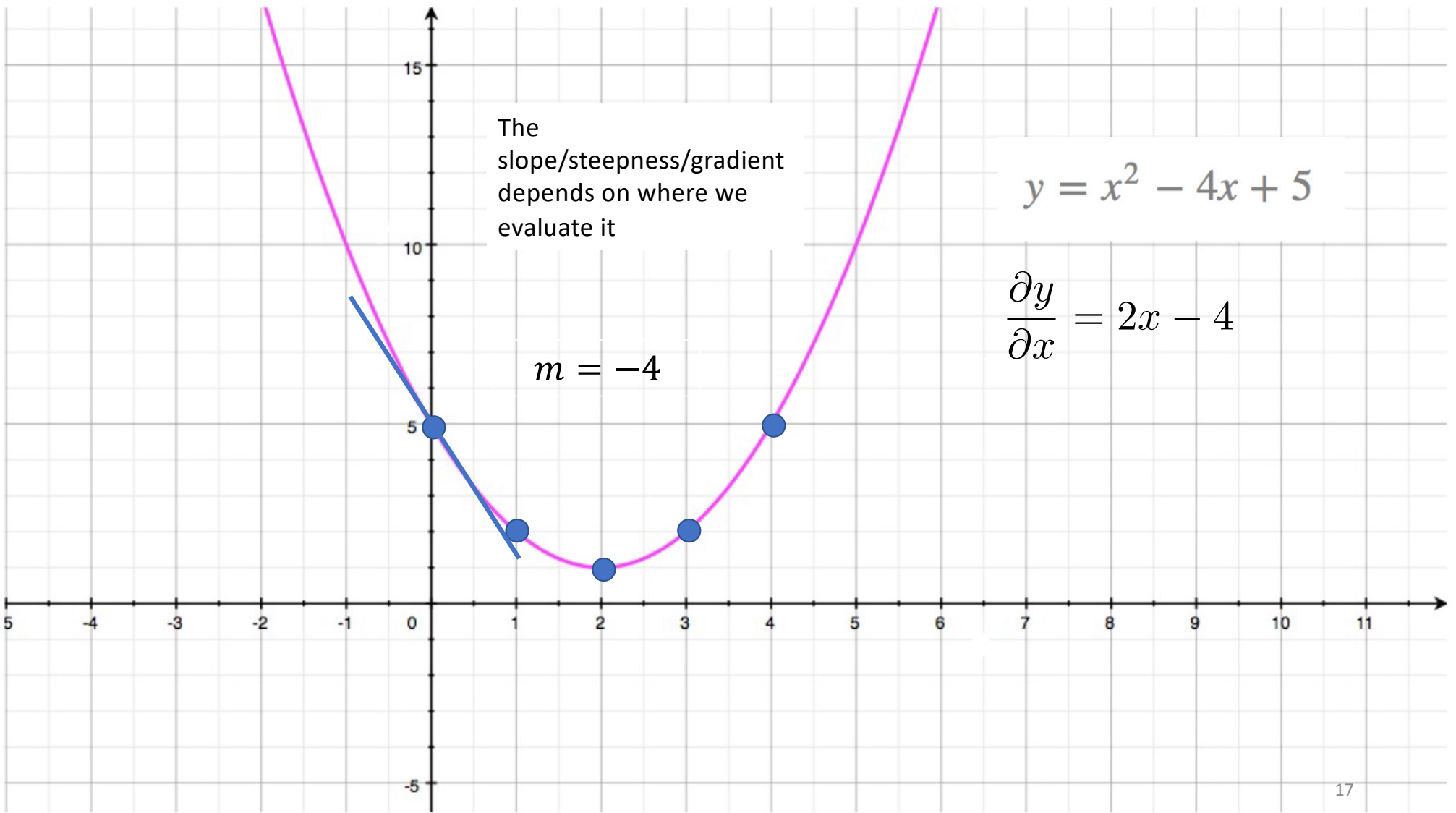








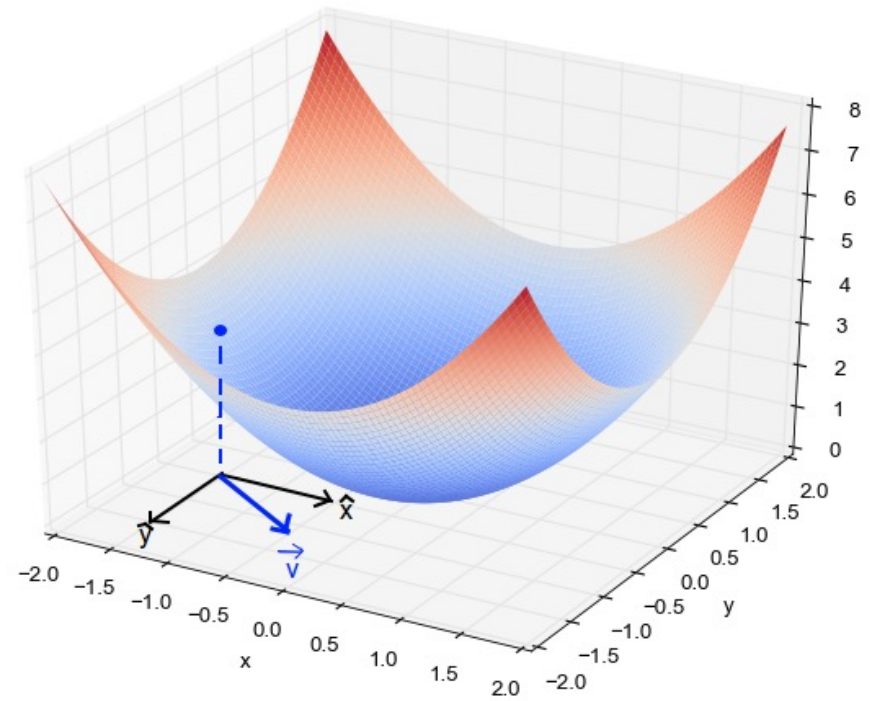




# Gradient

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

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)

# Fitting models

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

# 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:

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

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

# Fitting models

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

slido

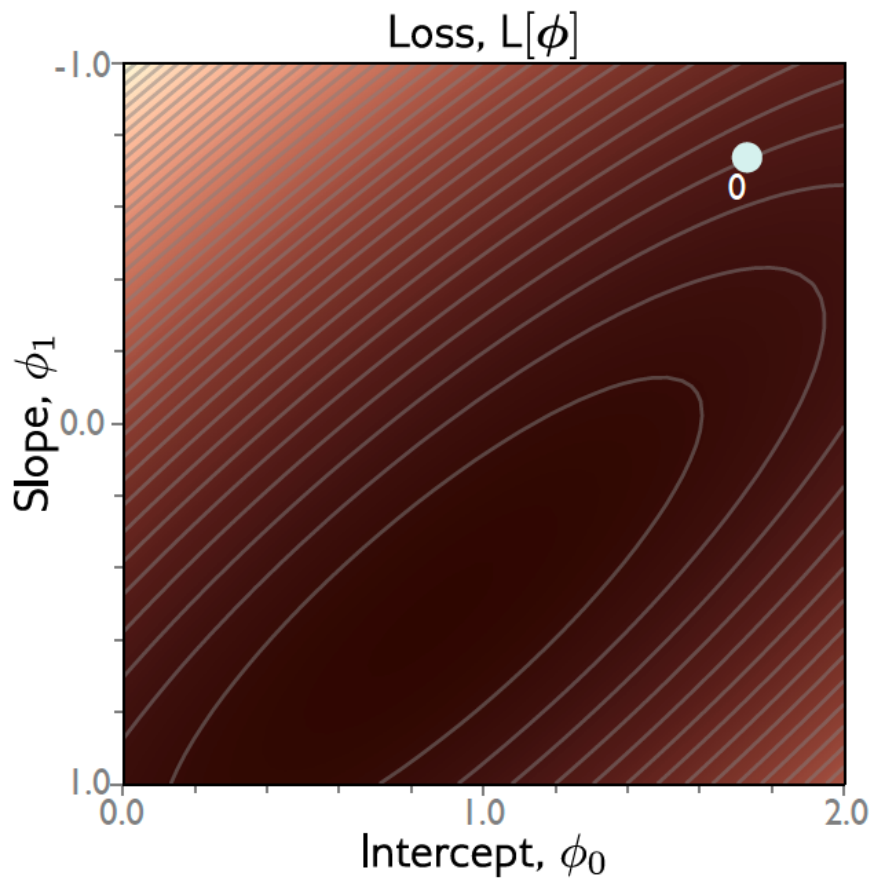
Please download and install the Slido app on all computers you use



**What does the gradient of a function represent in optimization?**

① Start presenting to display the poll results on this slide.

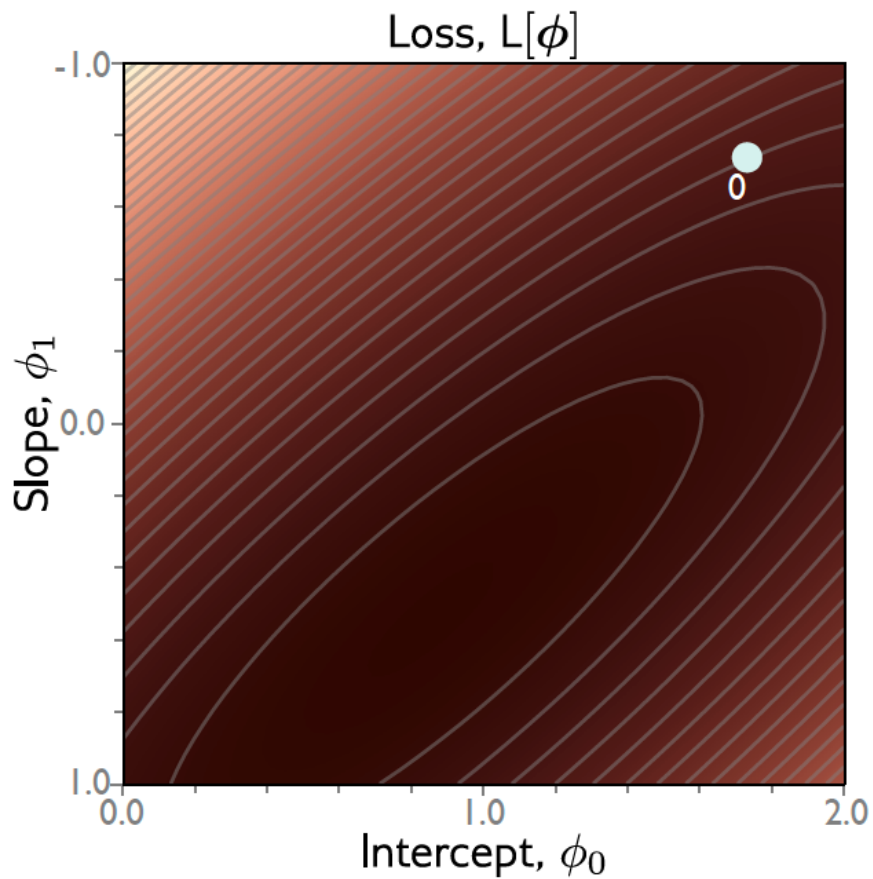
# Gradient descent



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



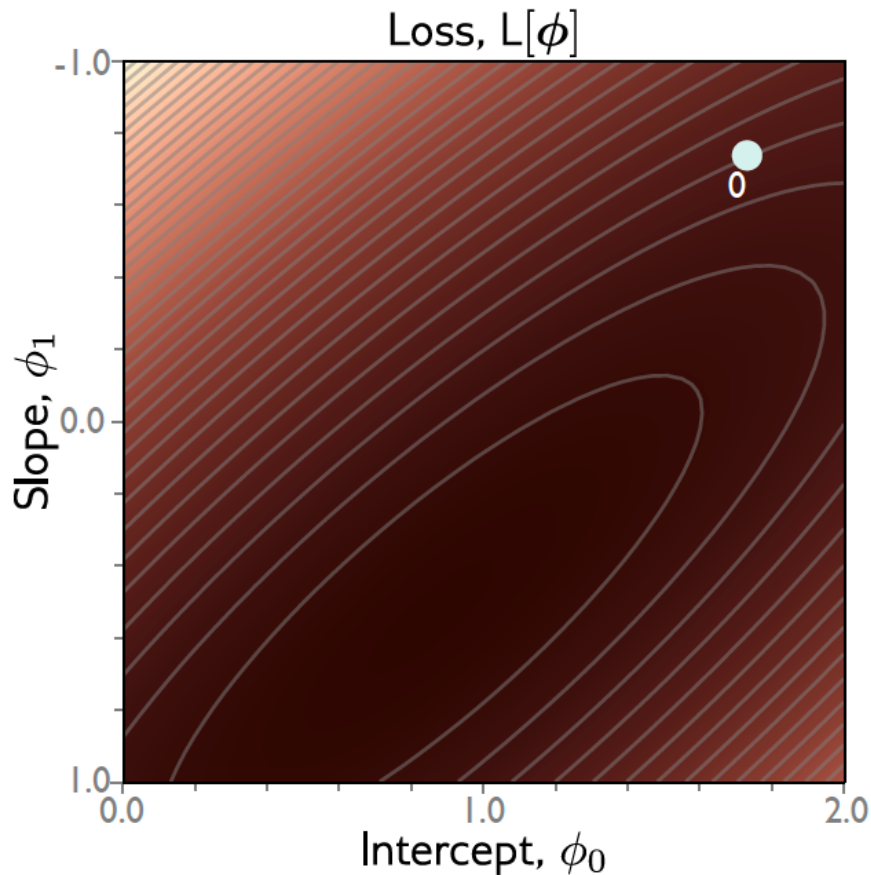
Step 1: Compute derivatives (slopes of function) with Respect to the parameters

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

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



# Gradient descent



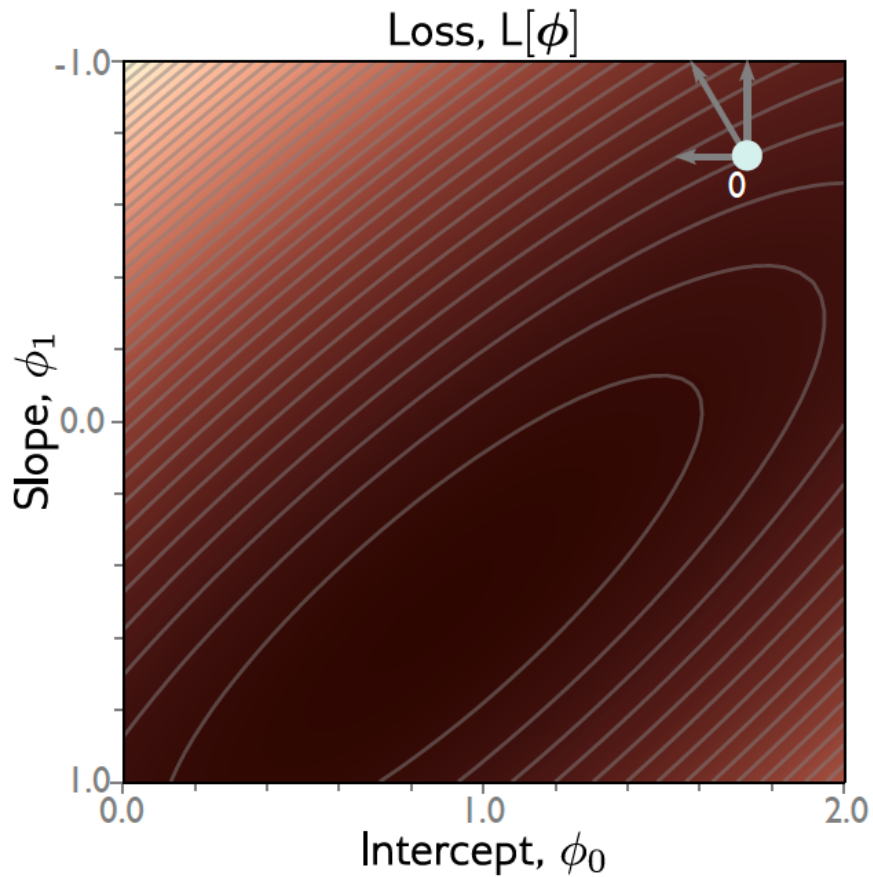
Step 1: Compute derivatives (slopes of function) with respect to the parameters

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

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

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

# Gradient descent

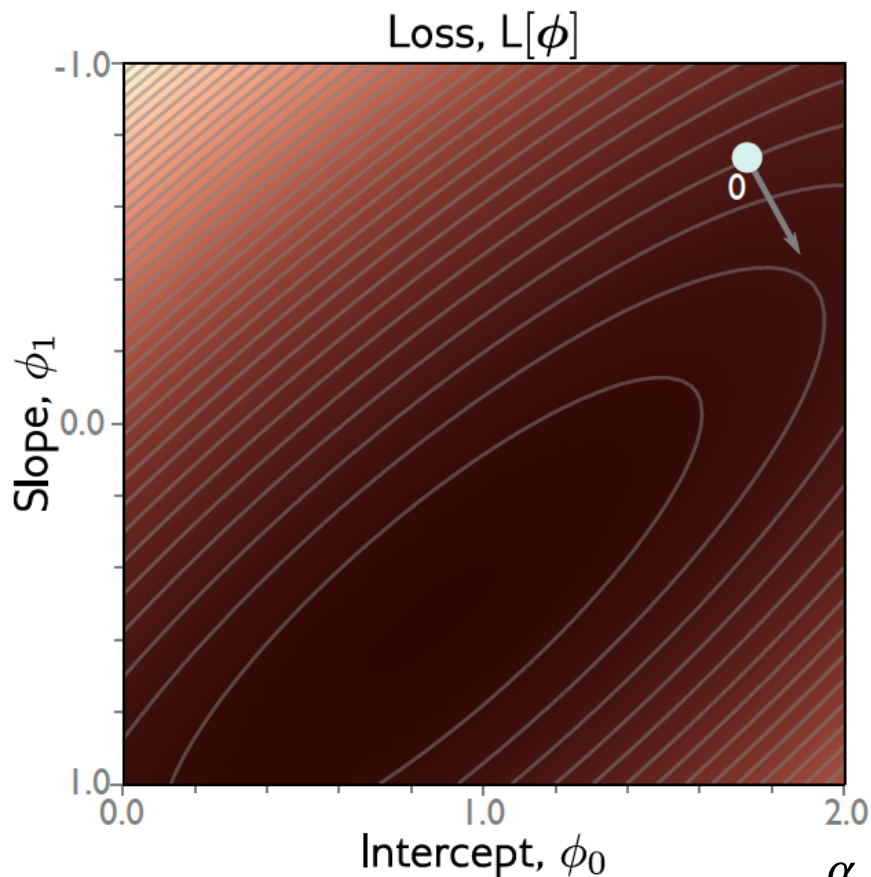


Step 1: Compute derivatives (slopes of function) with Respect to the parameters

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

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

# Gradient descent



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

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

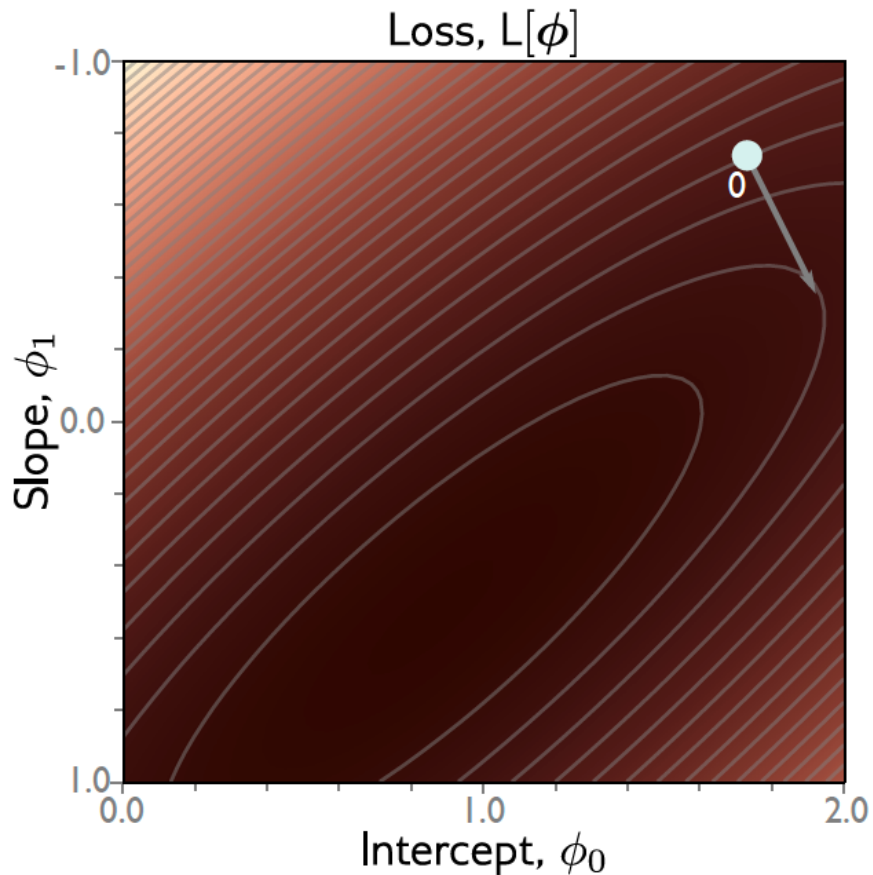
$$\frac{\partial \ell_i}{\partial \phi} = \begin{bmatrix} \frac{\partial \ell_i}{\partial \phi_0} \\ \frac{\partial \ell_i}{\partial \phi_1} \end{bmatrix} = \begin{bmatrix} 2(\phi_0 + \phi_1 x_i - y_i) \\ 2x_i(\phi_0 + \phi_1 x_i - y_i) \end{bmatrix}$$

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

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

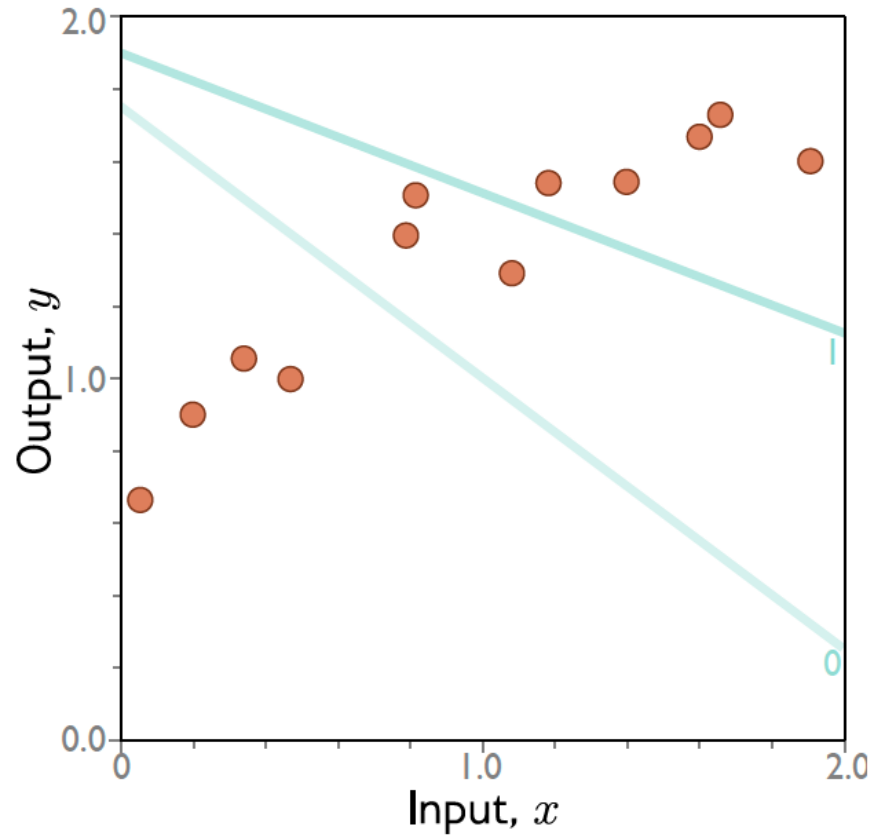
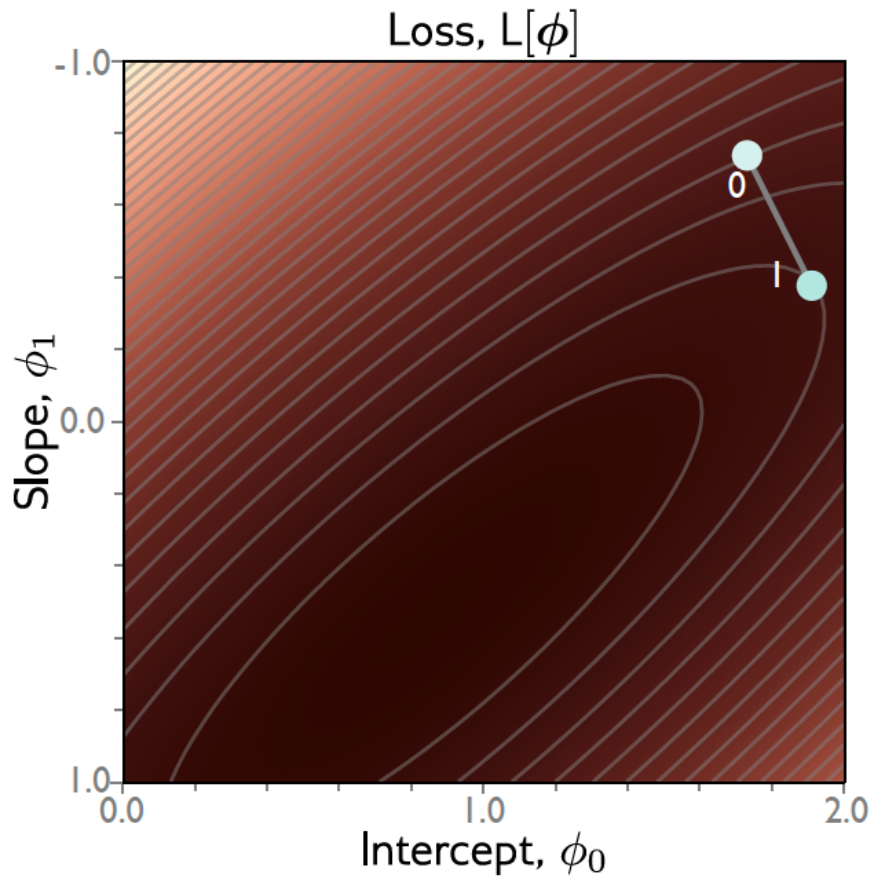
$$\frac{\partial \ell_i}{\partial \phi} = \begin{bmatrix} \frac{\partial \ell_i}{\partial \phi_0} \\ \frac{\partial \ell_i}{\partial \phi_1} \end{bmatrix} = \begin{bmatrix} 2(\phi_0 + \phi_1 x_i - y_i) \\ 2x_i(\phi_0 + \phi_1 x_i - y_i) \end{bmatrix}$$

Step 2: Update parameters according to rule

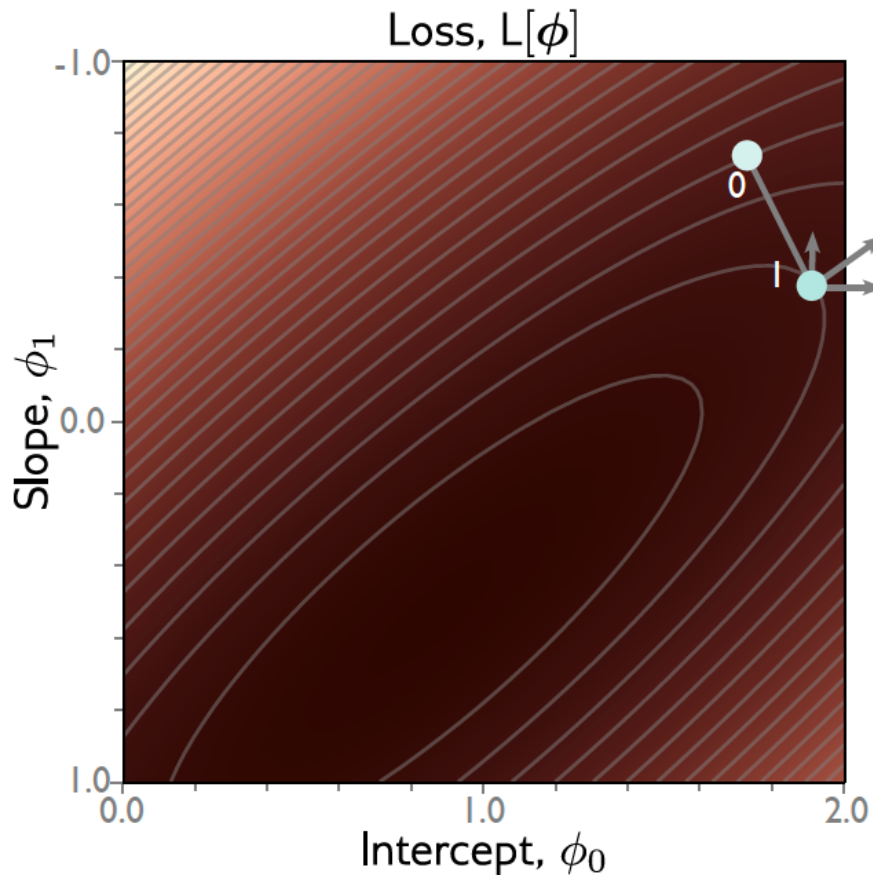
$$\phi \longleftarrow \phi - \alpha \frac{\partial L}{\partial \phi}$$

$\alpha$  = step size

# Gradient descent



# Gradient descent



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

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

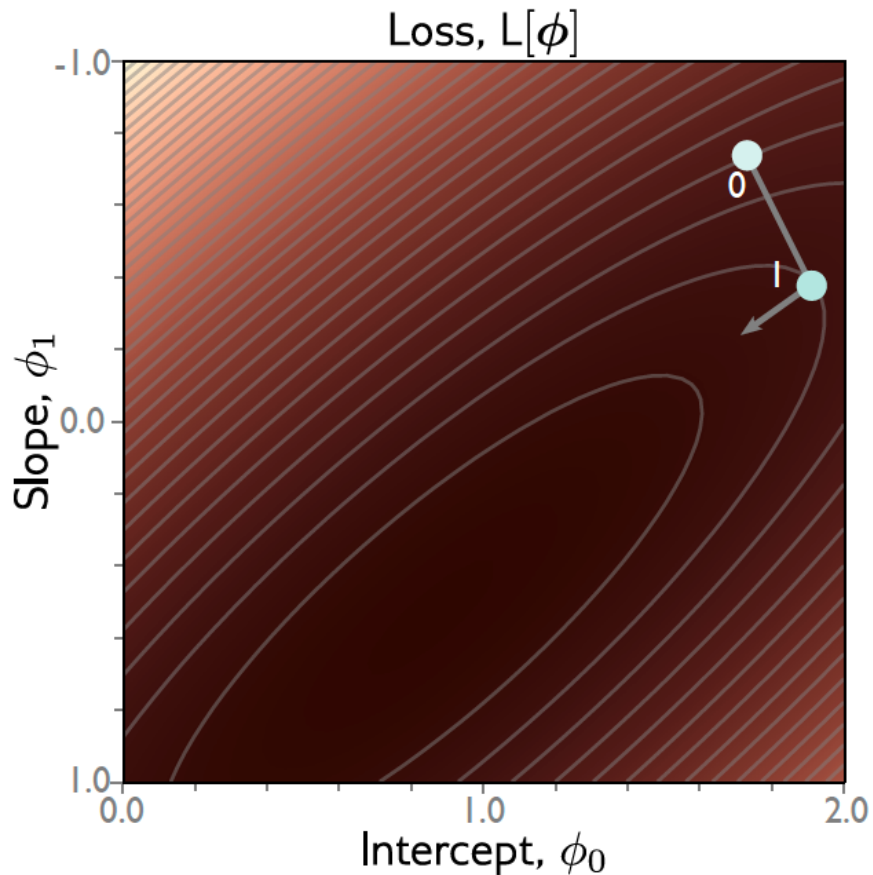
$$\frac{\partial \ell_i}{\partial \phi} = \begin{bmatrix} \frac{\partial \ell_i}{\partial \phi_0} \\ \frac{\partial \ell_i}{\partial \phi_1} \end{bmatrix} = \begin{bmatrix} 2(\phi_0 + \phi_1 x_i - y_i) \\ 2x_i(\phi_0 + \phi_1 x_i - y_i) \end{bmatrix}$$

Step 2: Update parameters according to rule

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

$\alpha$  = step size

# Gradient descent



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

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

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

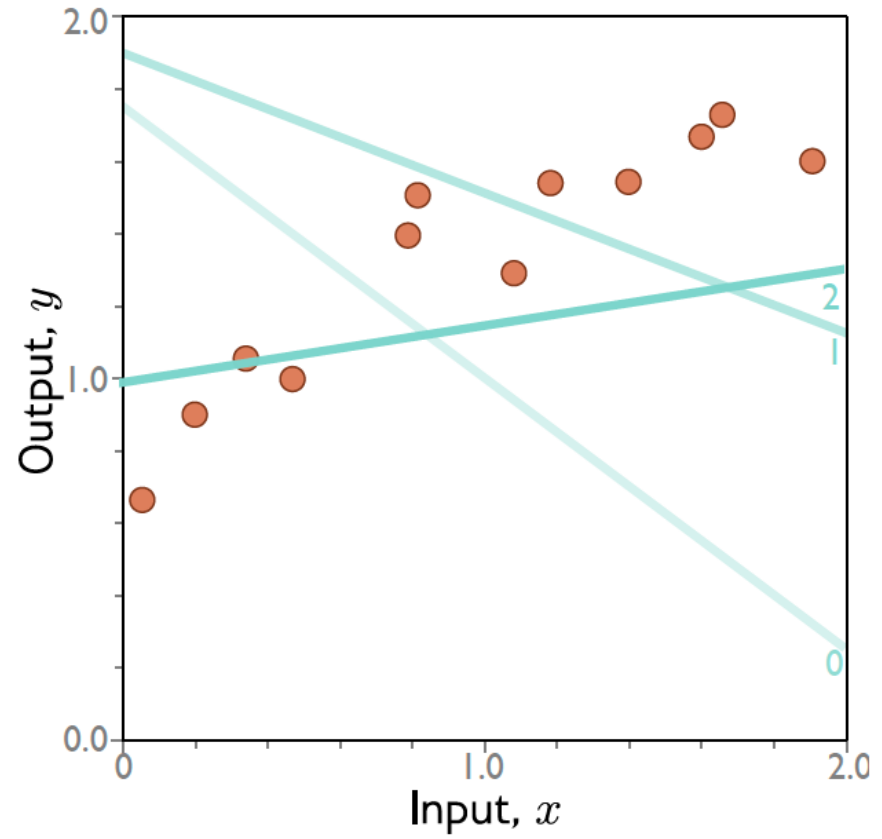
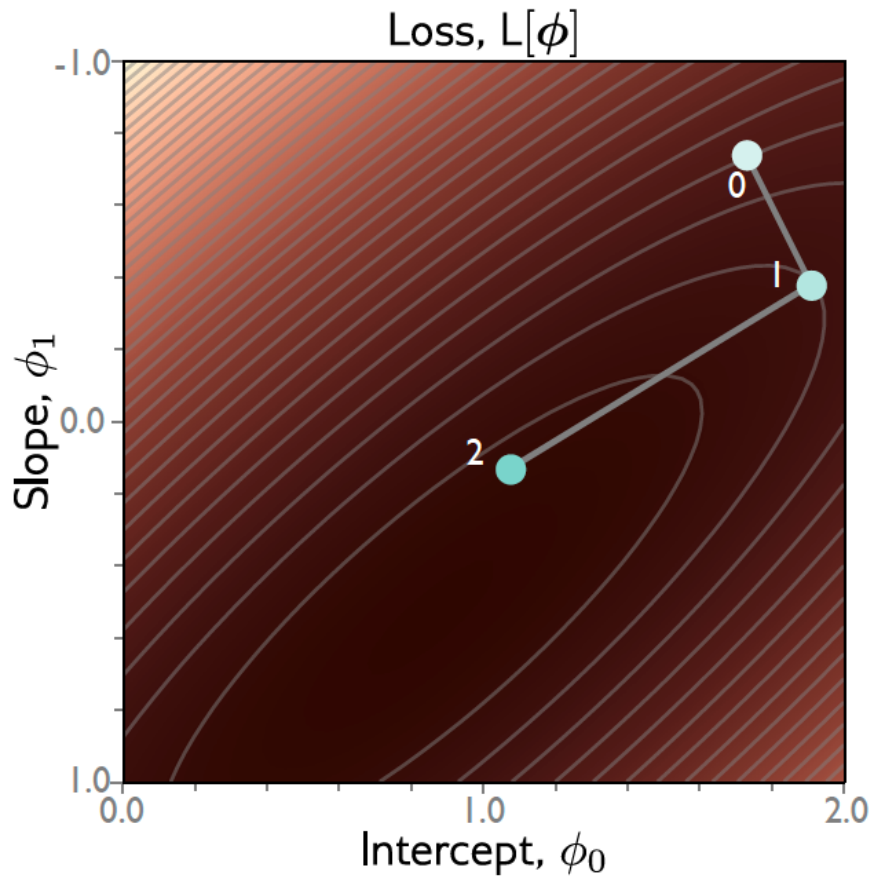
Step 2: Update parameters according to rule

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

$\alpha$  = step size

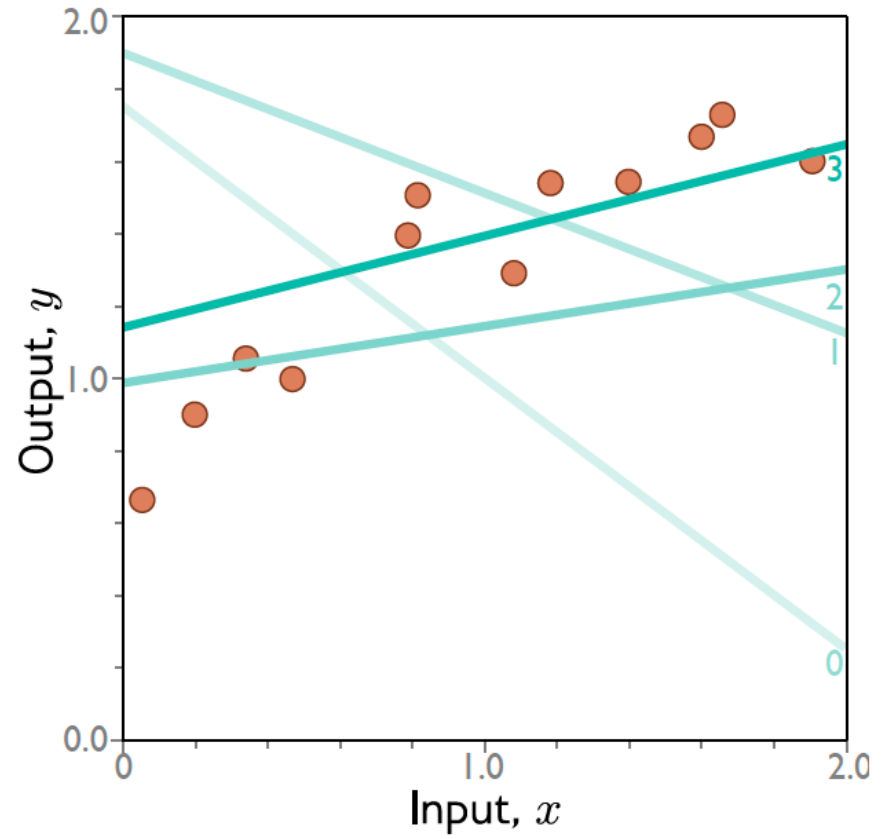
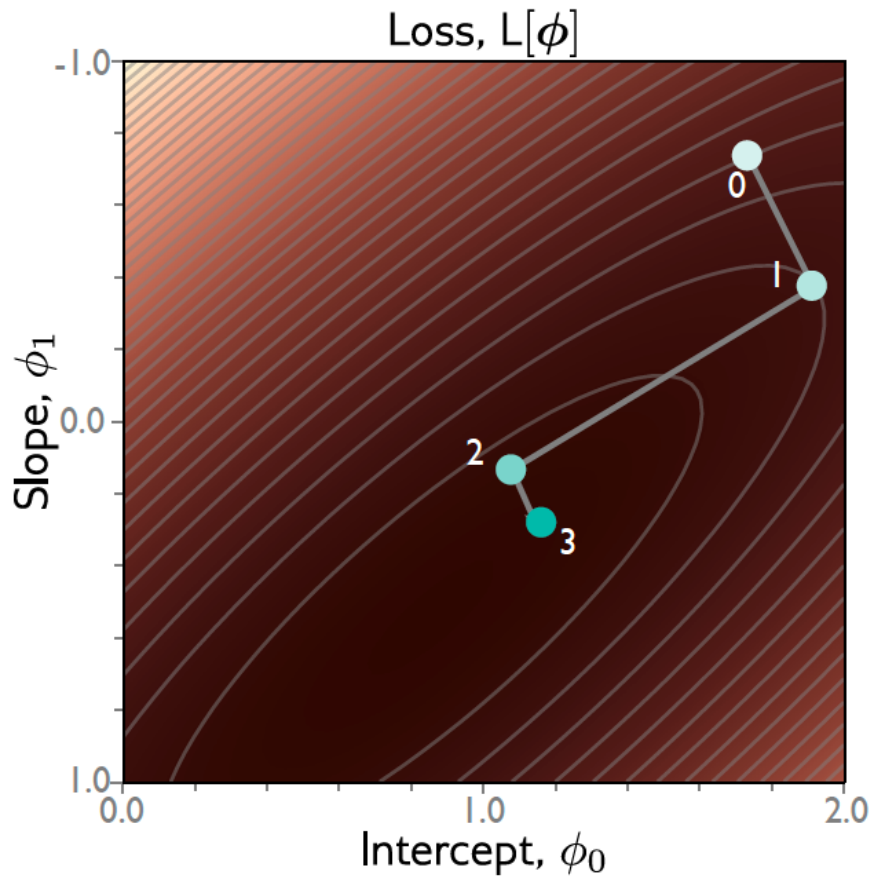


# Gradient descent

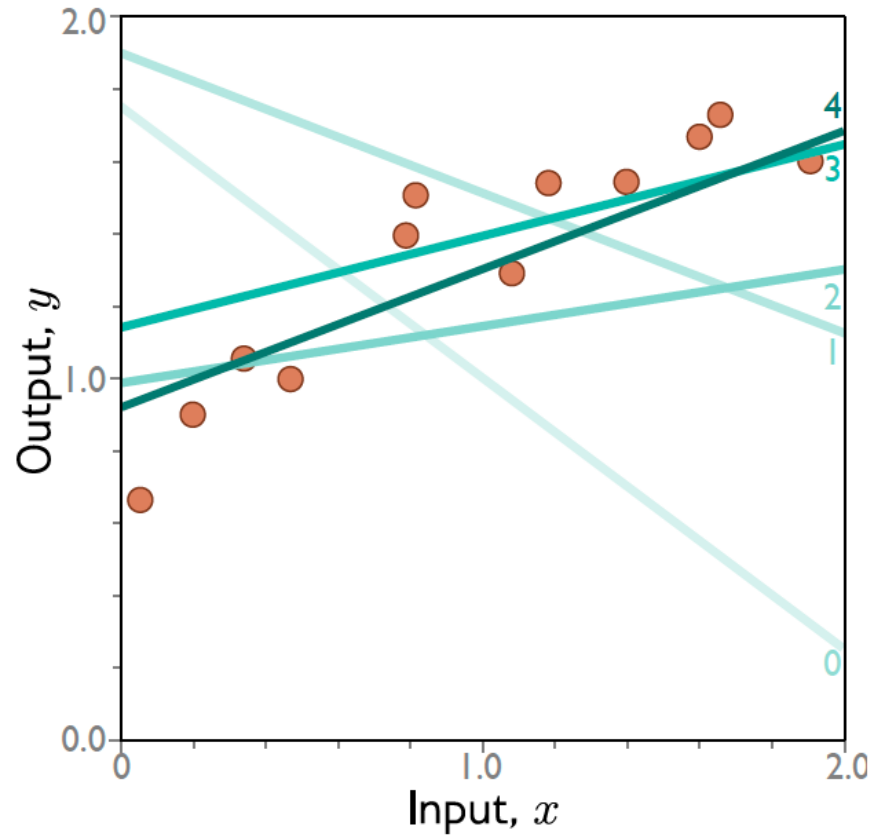
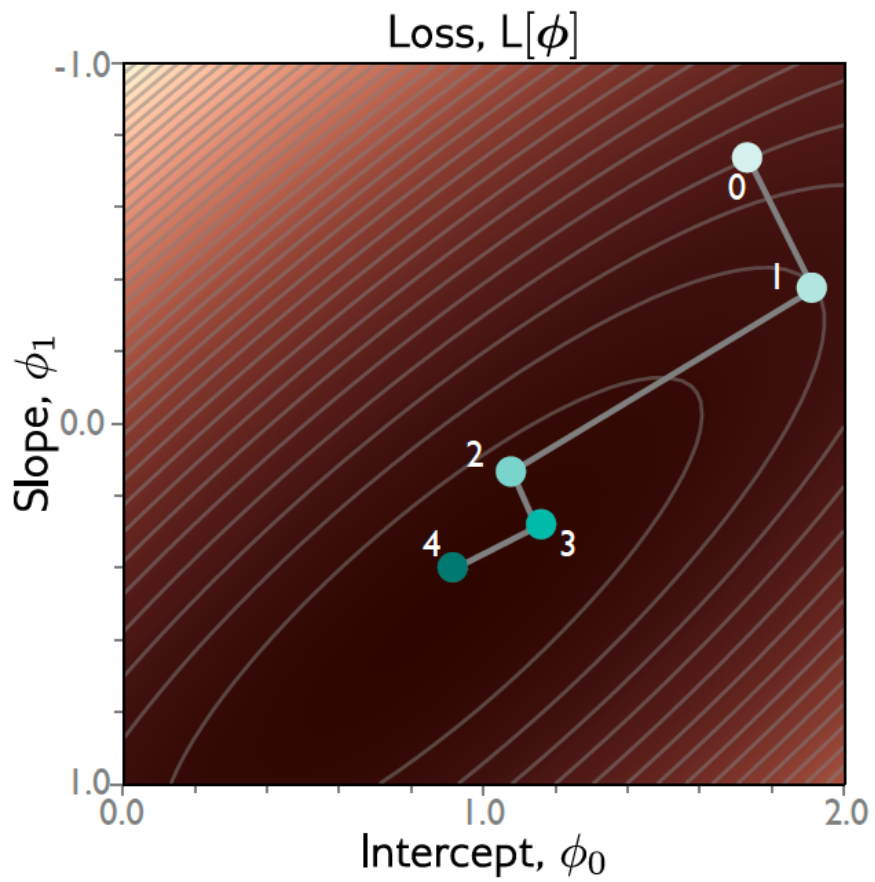




# Gradient descent



# Gradient descent



**slido**

Please download and install the Slido app on all computers you use



**What is the effect of a learning rate that is too large?**

① Start presenting to display the poll results on this slide.

# Fitting models

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

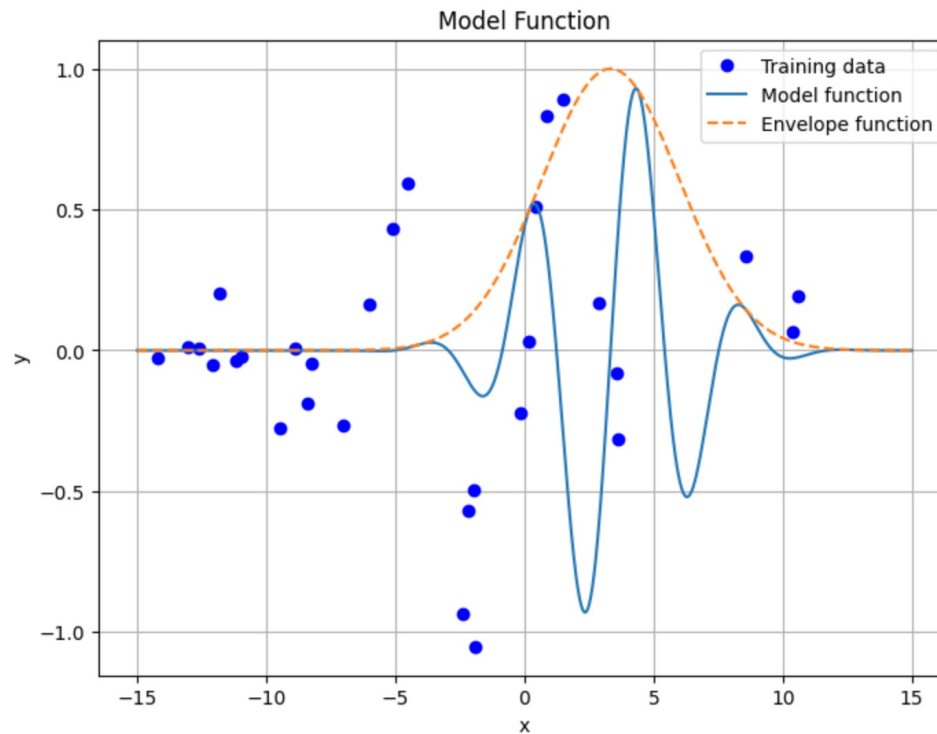
The linear model loss function was convex.

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

➔ Gabor Function

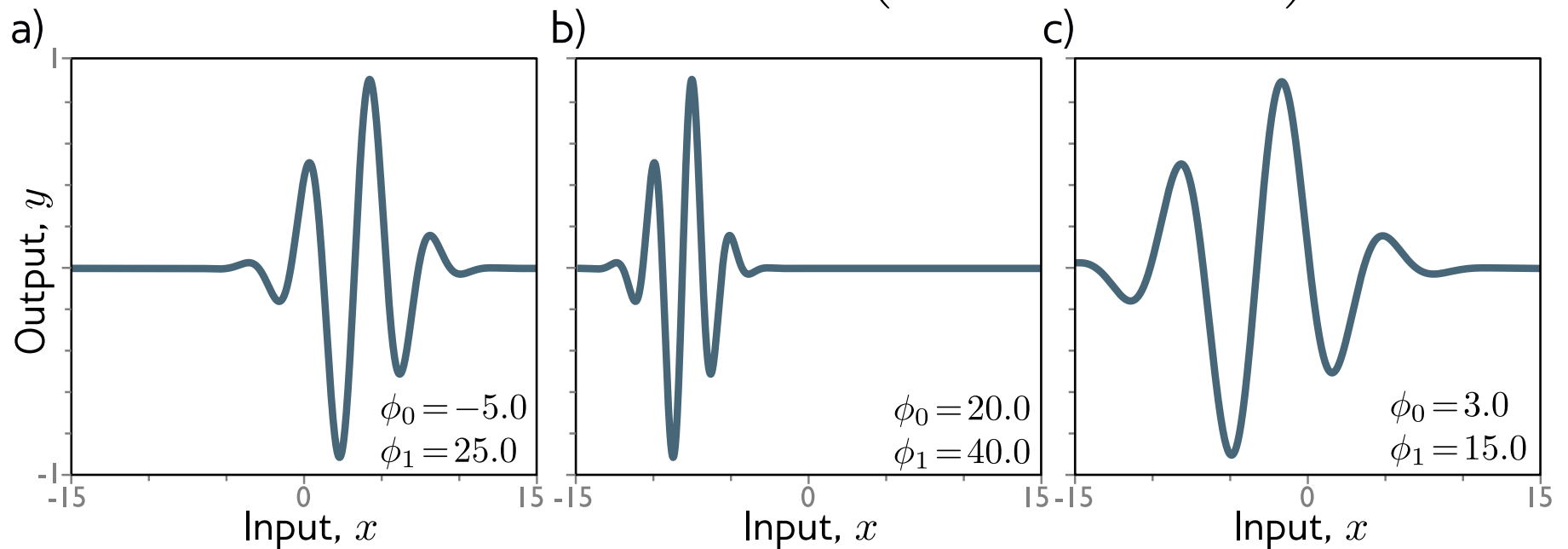
# Gabor Model (with Envelope)

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



# Gabor model

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

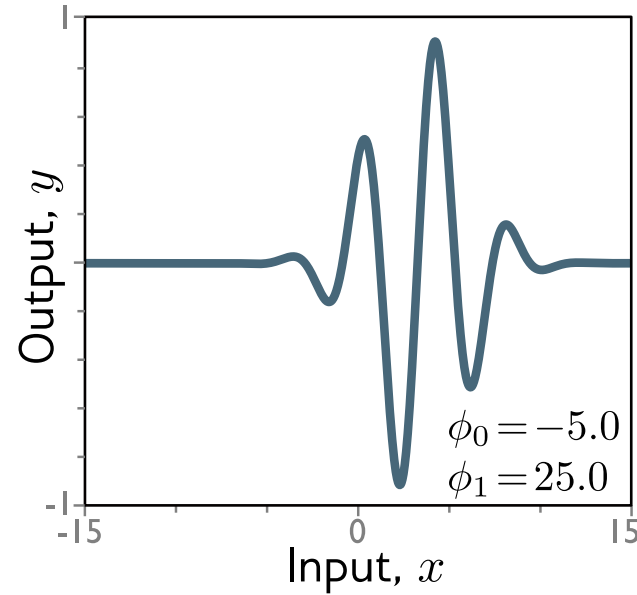
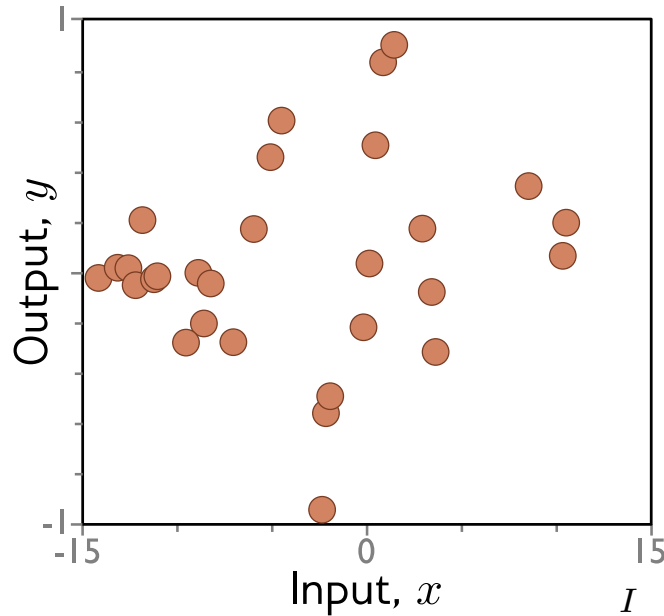


$\phi_0$  shifts left and right

$\phi_1$  shrinks and expands the sinusoid and envelope

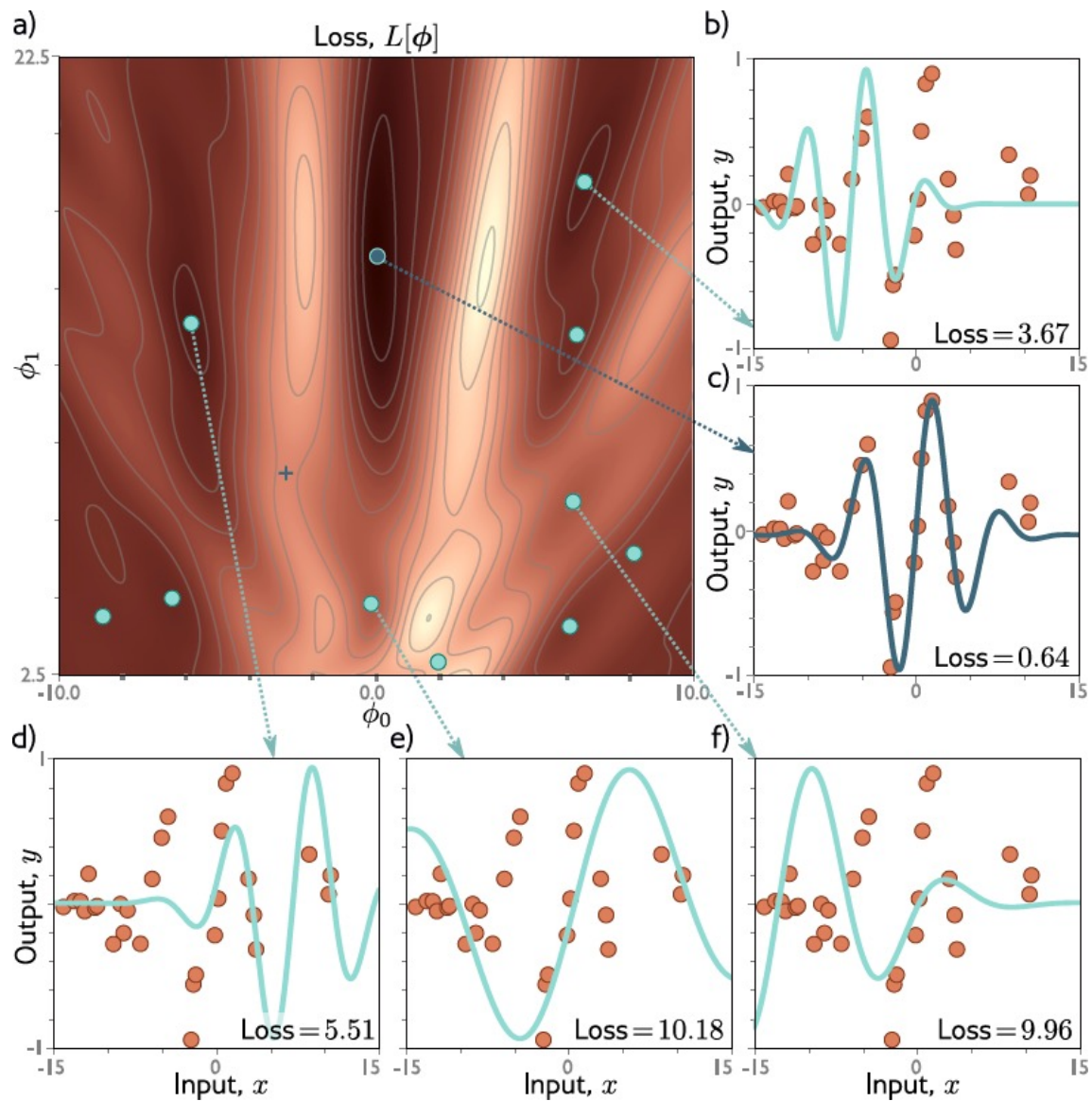
# Toy Dataset and Gabor model

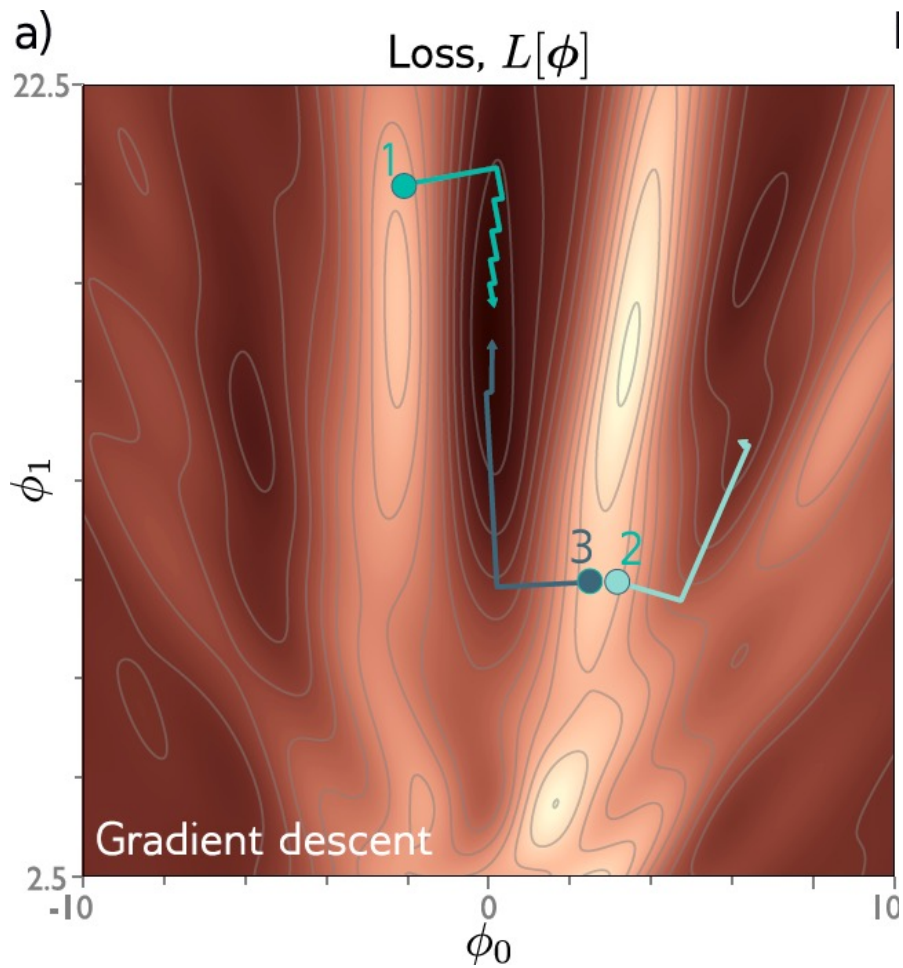
$$f[x, \phi] = \sin[\phi_0 + 0.06 \cdot \phi_1 x] \cdot \exp\left(-\frac{(\phi_0 + 0.06 \cdot \phi_1 x)^2}{8.0}\right)$$



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



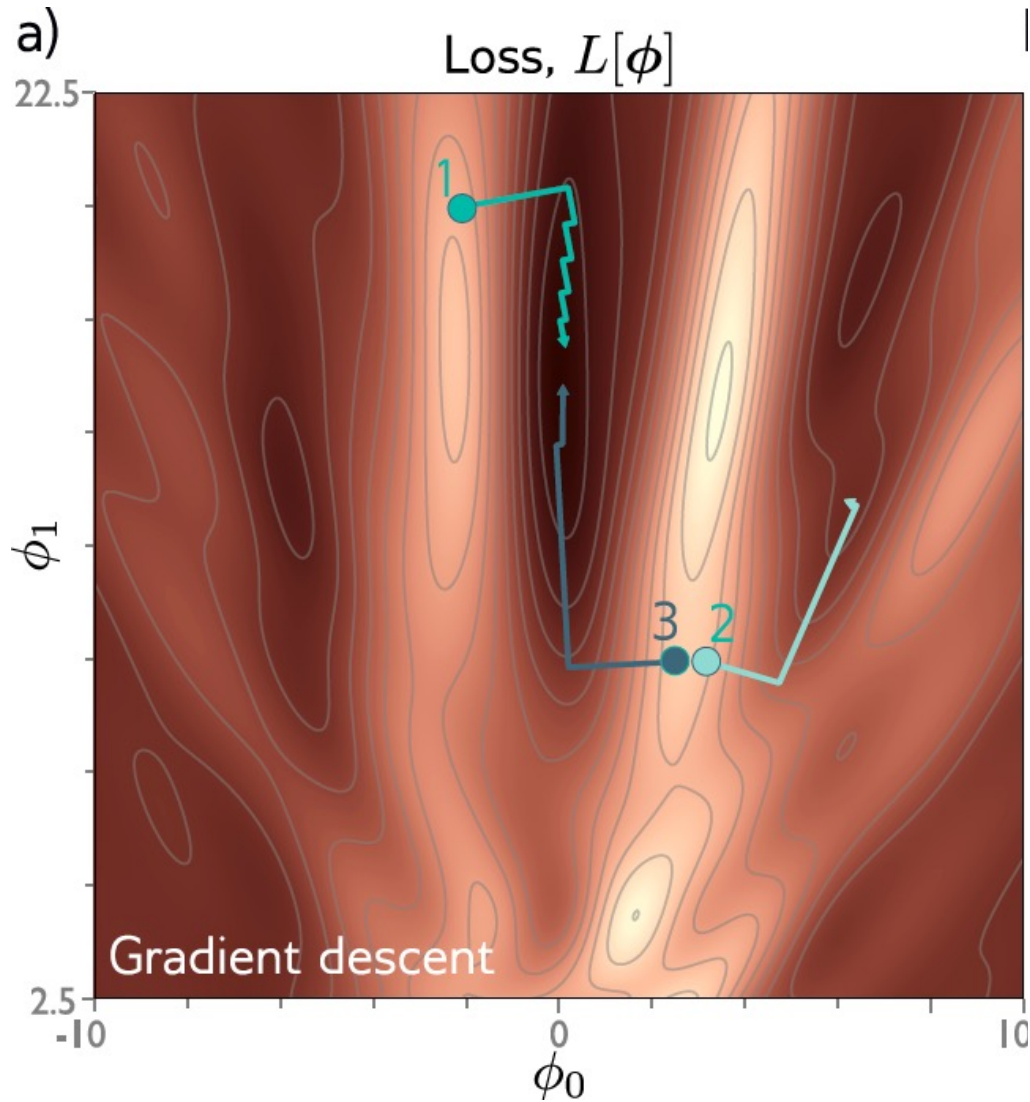




- Gradient descent gets to the global minimum if we start in the right “valley”
- Otherwise, descends to a local minimum
- Or get stuck near a saddle point

# Fitting models

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

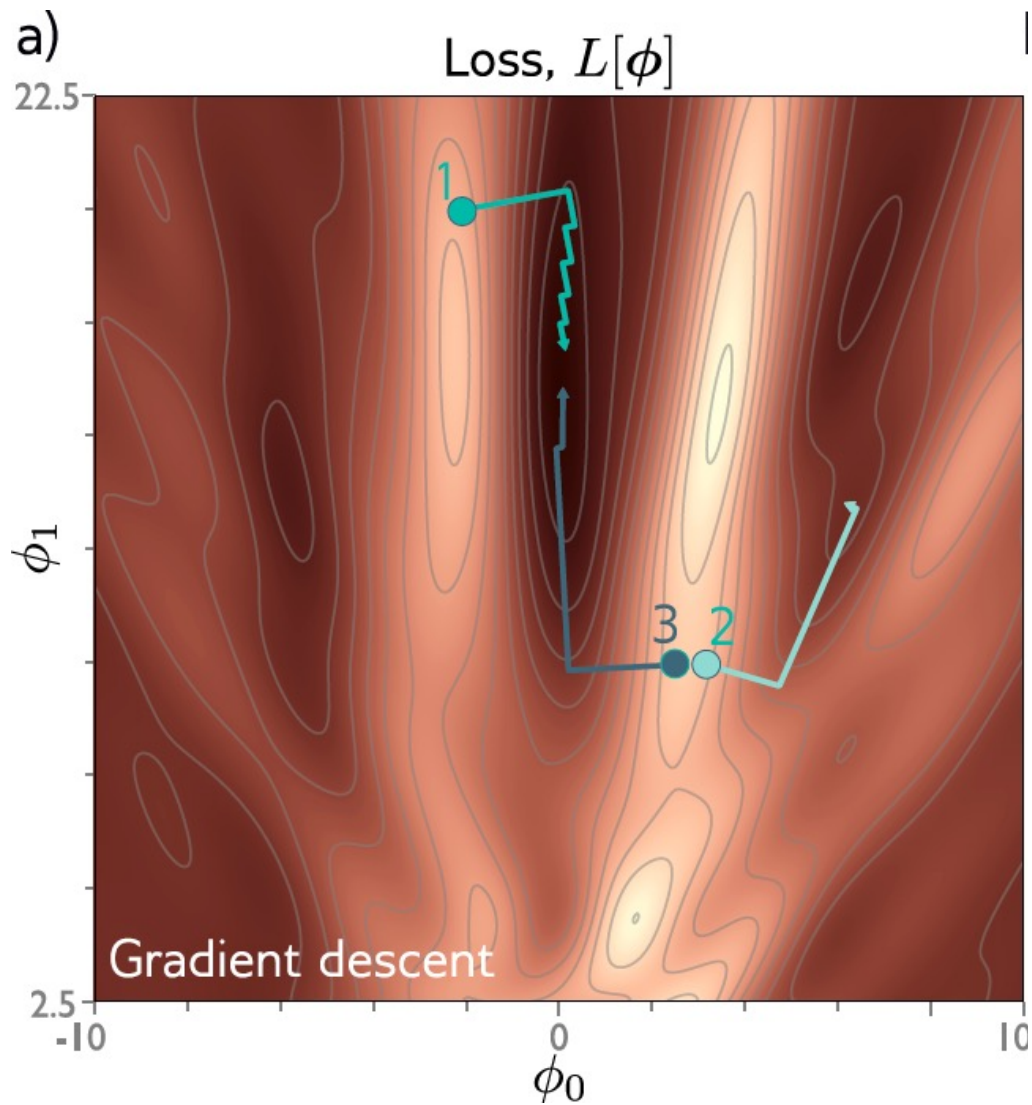
(Ex. 30 sample dataset, batch size 5)

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)

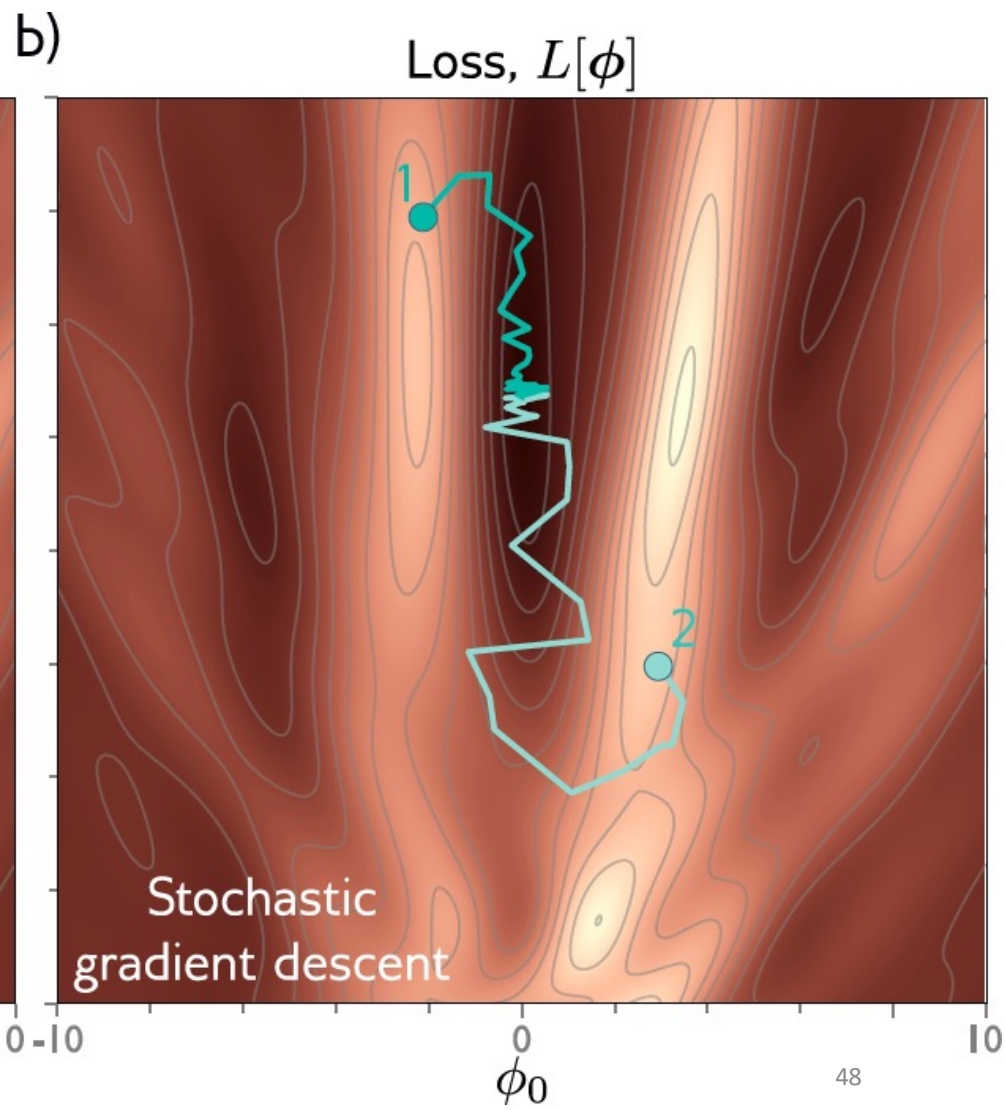
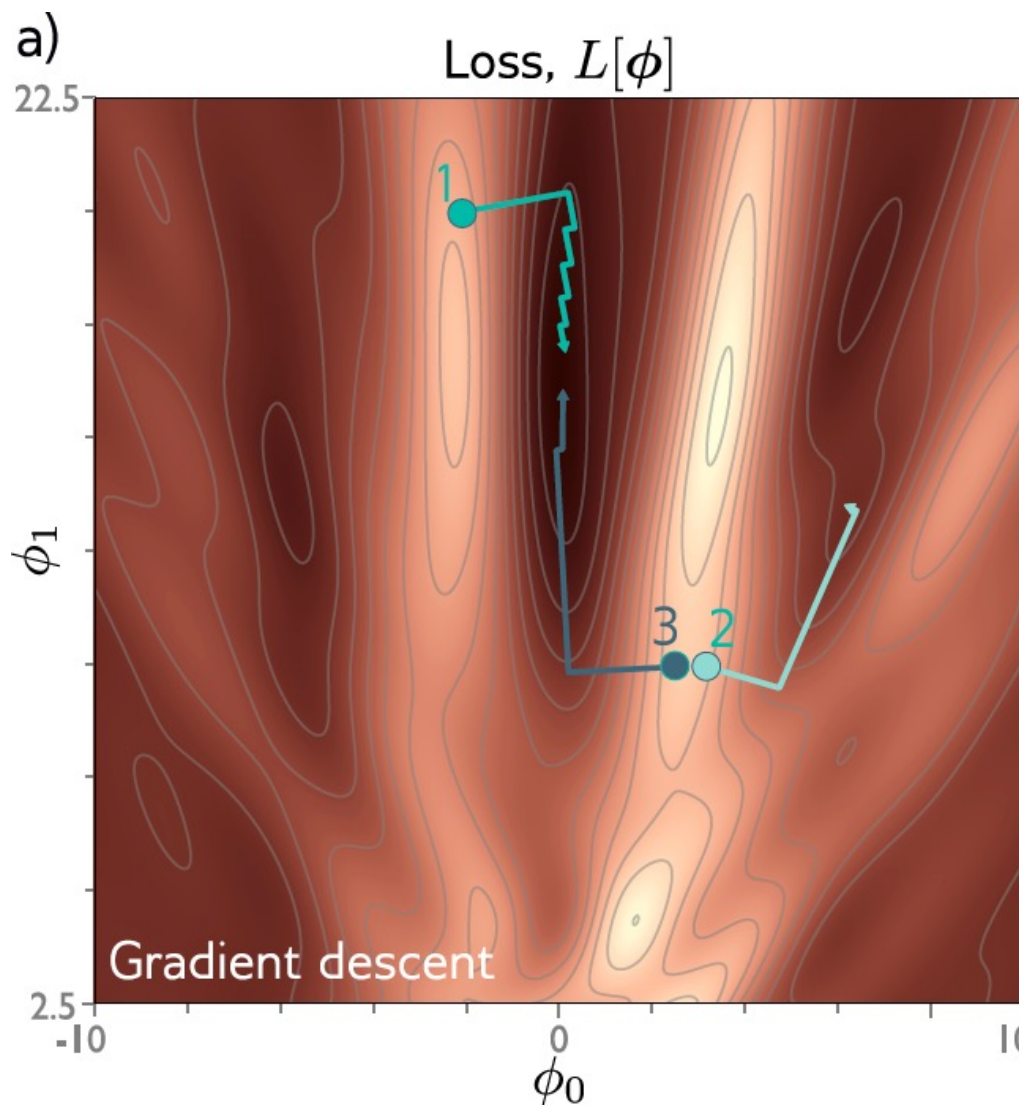
$$\phi_{t+1} \leftarrow \phi_t - \alpha \sum_{i=1}^I \frac{\partial l_i[\phi_t]}{\partial \phi},$$

After (SGD)

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

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



slido

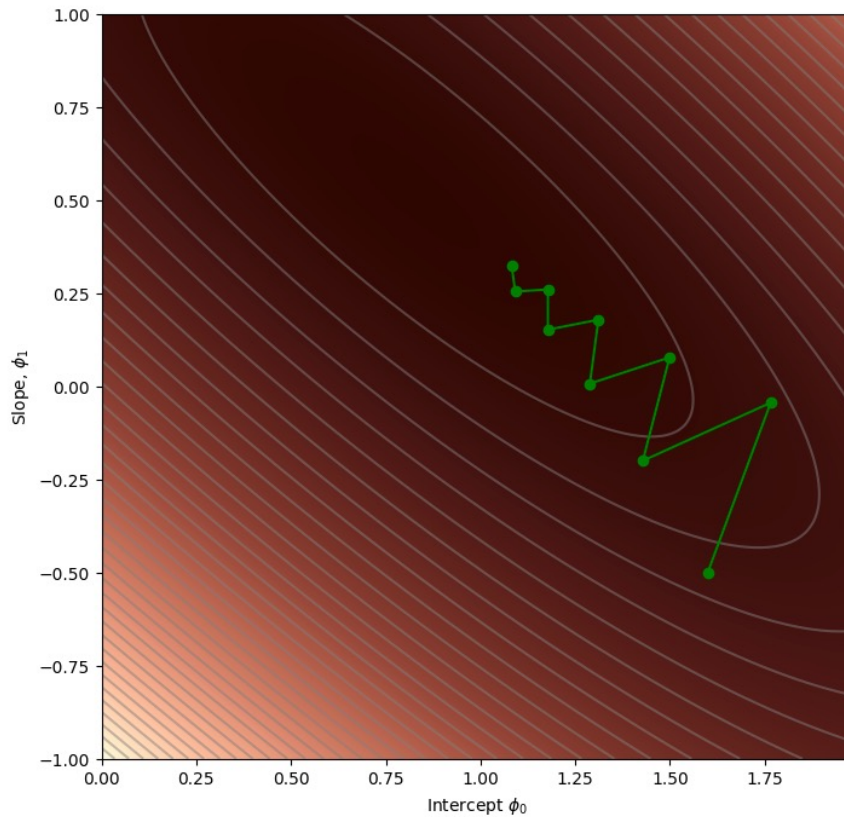
Please download and install the Slido app on all computers you use



**Which of the following is an advantage of SGD over full-batch gradient descent?**

① Start presenting to display the poll results on this slide.

# 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?


# Fitting models

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

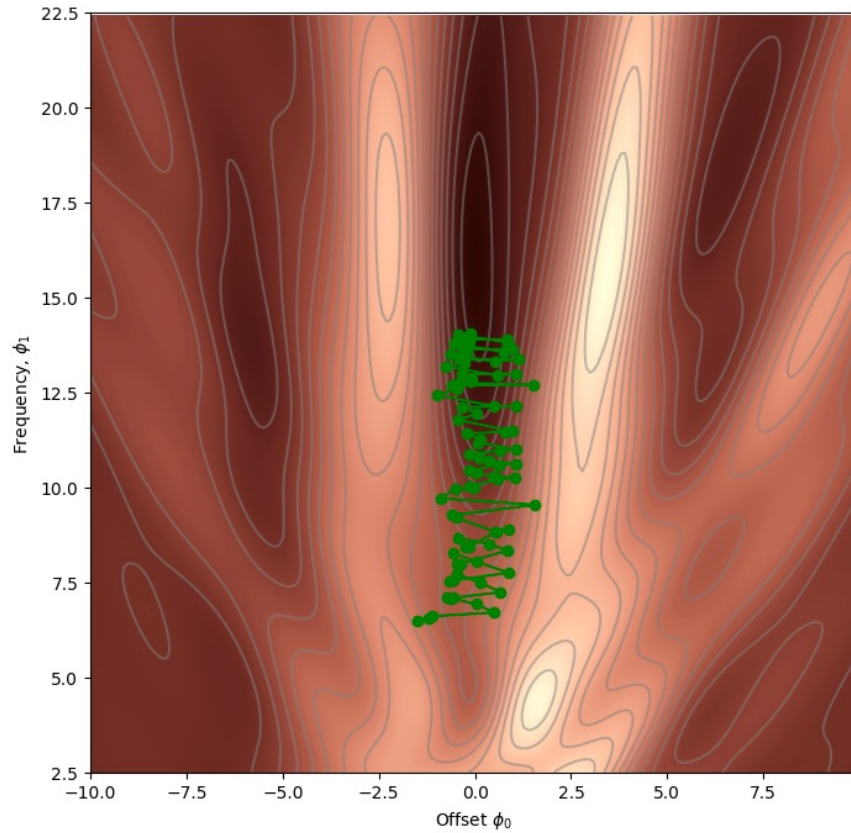
# Momentum

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

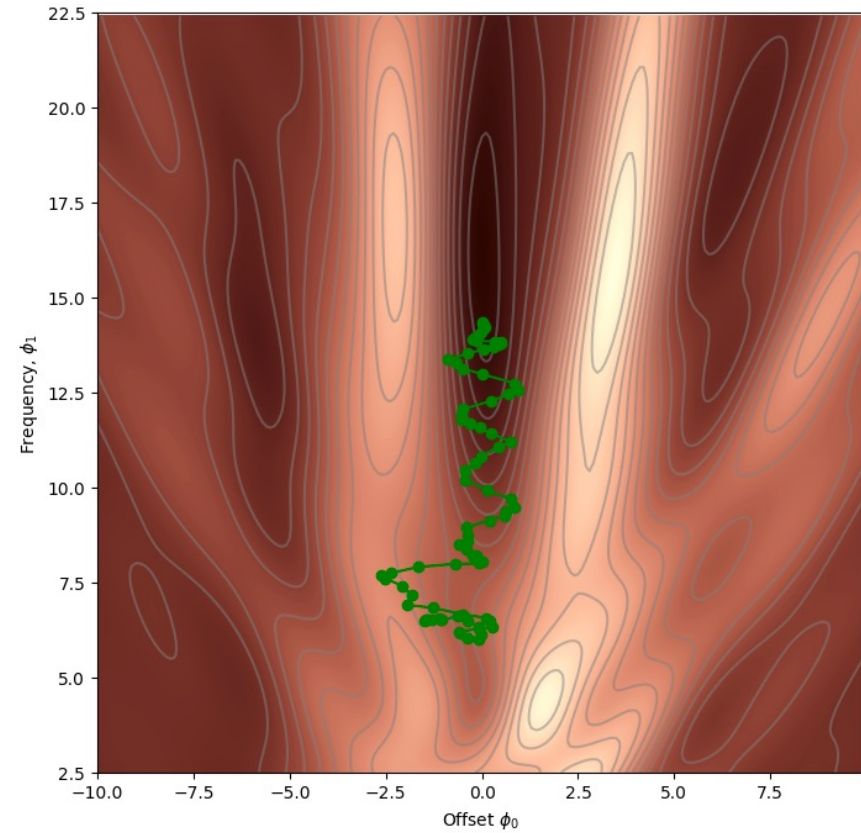
$$\mathbf{m}_{t+1} \leftarrow \beta \cdot \mathbf{m}_t + (1 - \beta) \sum_{i \in \mathcal{B}_t} \frac{\partial \ell_i[\phi_t]}{\partial \phi}$$
$$\phi_{t+1} \leftarrow \phi_t - \alpha \cdot \mathbf{m}_{t+1}$$

 Still in batches.

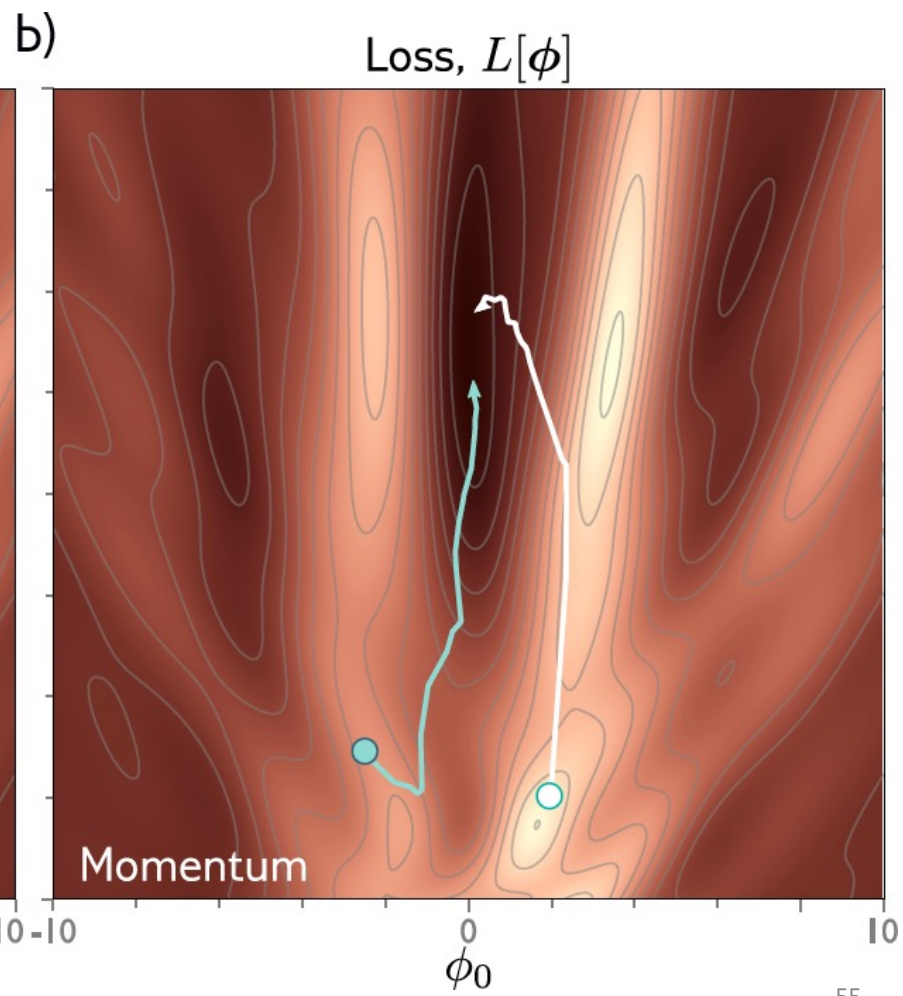
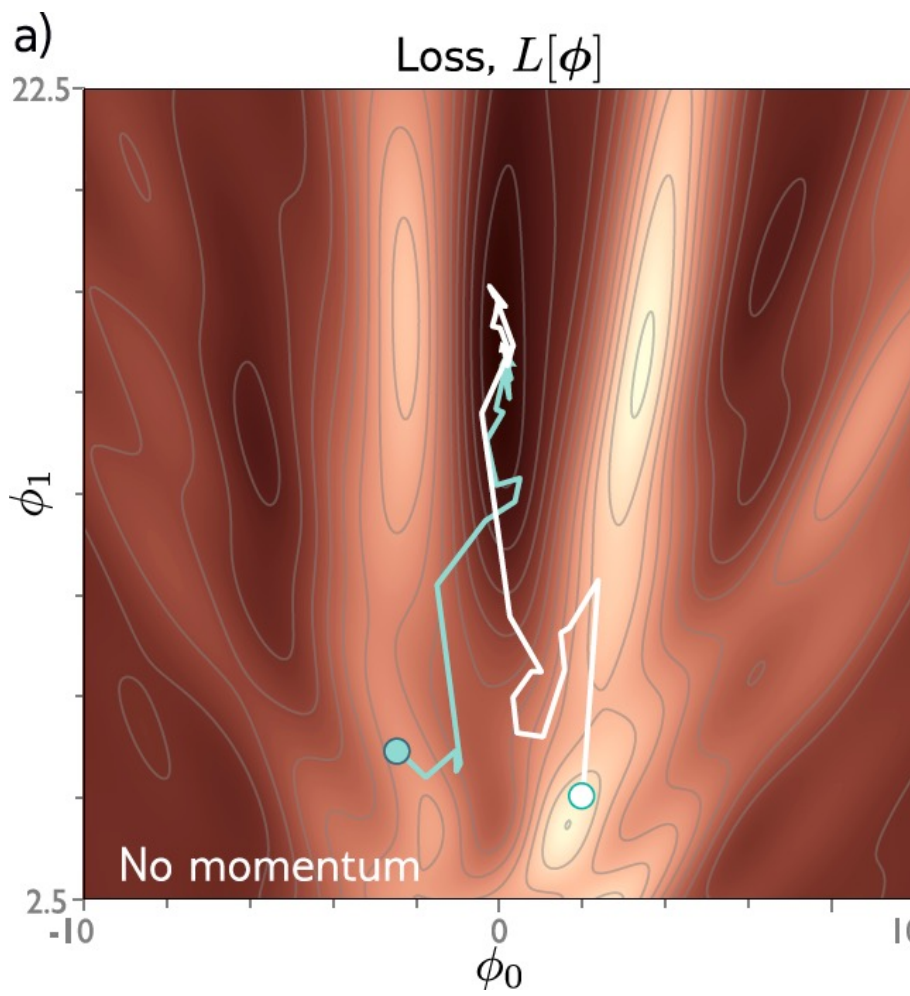
# Without and With Momentum



Without Momentum, Loss = 1.31



With Momentum, Loss = 0.96



# Nesterov accelerated momentum

- Momentum smooths out gradient of current location

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

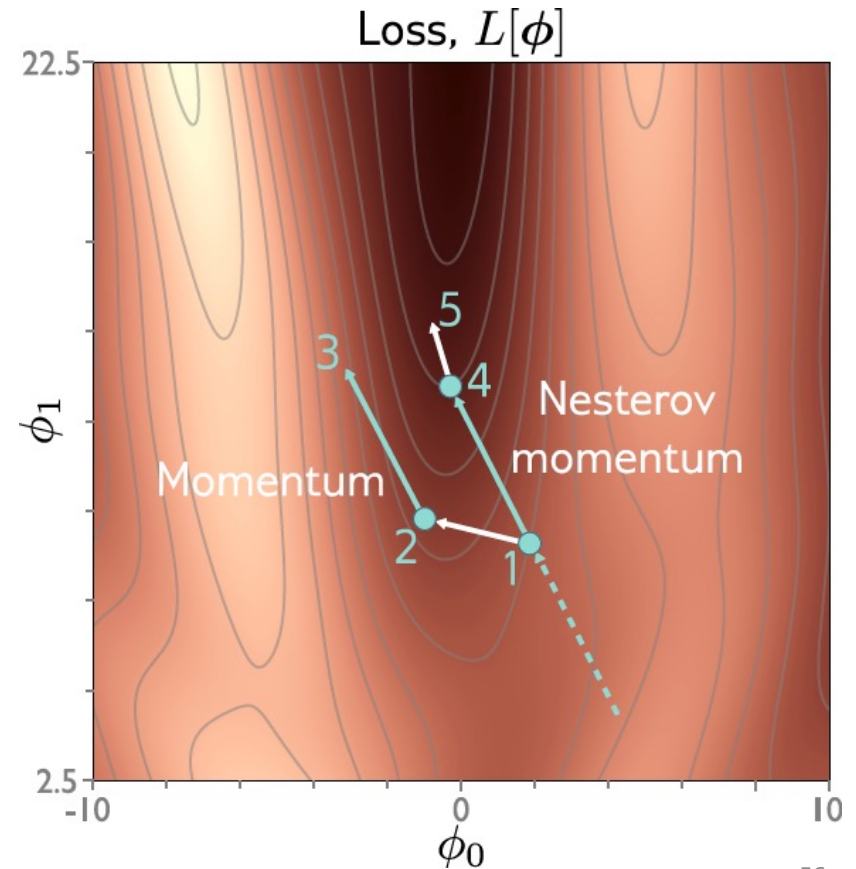
$$\phi_{t+1} \leftarrow \phi_t - \alpha \cdot \mathbf{m}_{t+1}$$

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

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

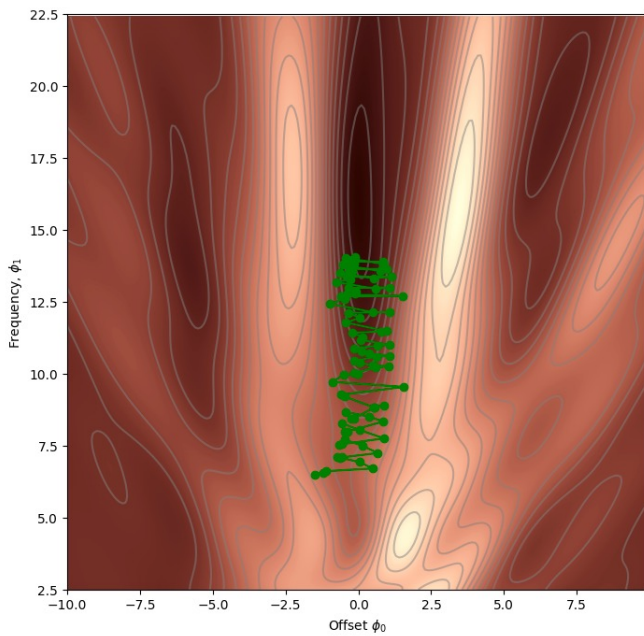
$$\phi_{t+1} \leftarrow \phi_t - \alpha \cdot \mathbf{m}_{t+1}$$

Still in batches.

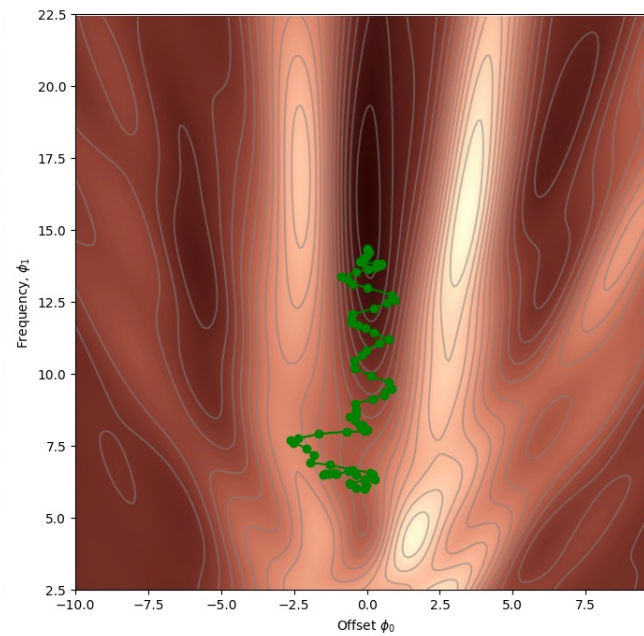




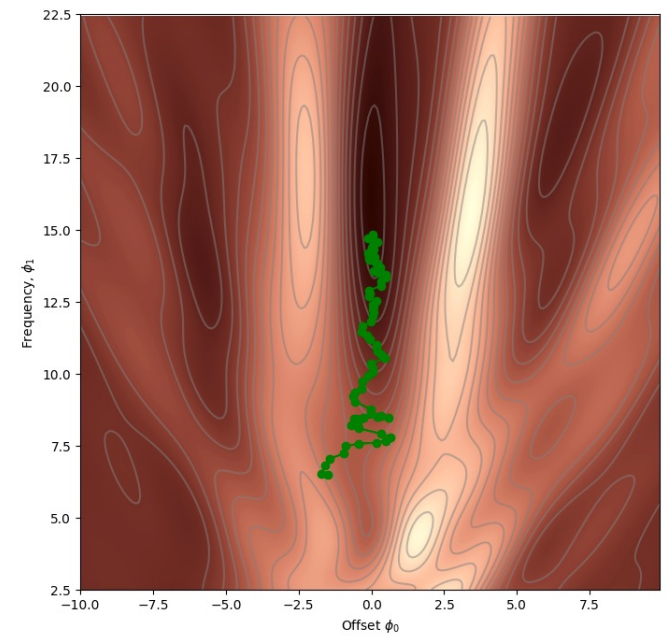
# Nesterov Momentum



Without Momentum, Loss = 1.31



With Momentum, Loss = 0.96



Nesterov Momentum, Loss = 0.80

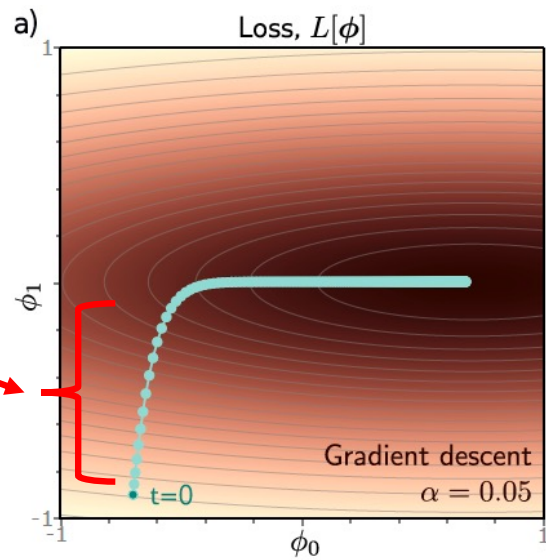


# Fitting models

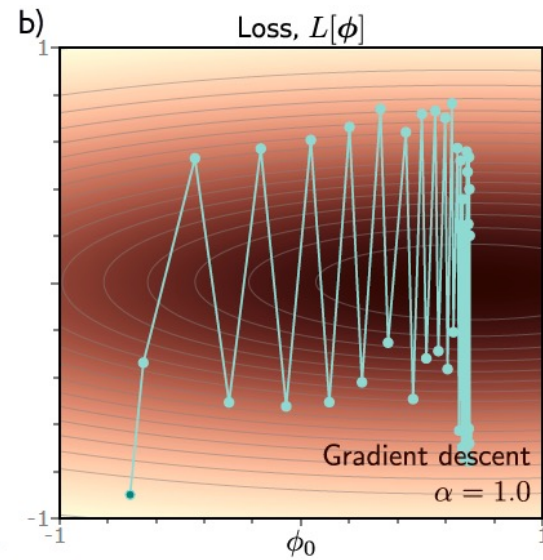
- Maths overview
- Gradient descent algorithm
- Linear regression example
- Gabor model example
- Stochastic gradient descent
- Momentum
- Adam

# The challenge with fixed step sizes

Moves quickly in one dimension but slowly in the other.



Too small and it will converge slowly, but eventually get there.



Too big and it will move quickly but might bounce around minimum or away.

# Solution Part 1: 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 1: 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 1: Normalized gradients

- Measure mean and pointwise squared gradient

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

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

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

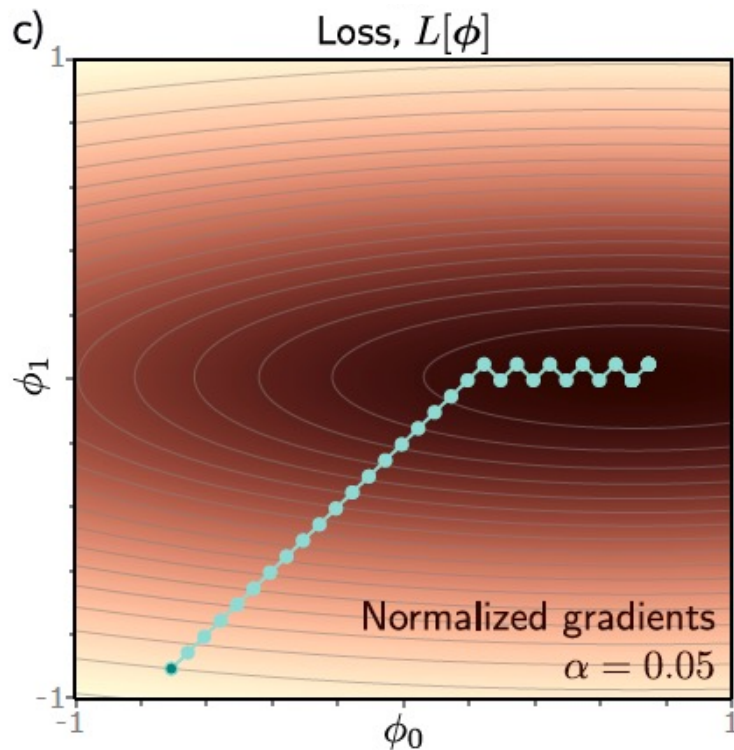
$$\mathbf{v}_{t+1} = \begin{bmatrix} 9.0 \\ 4.0 \\ 25.0 \end{bmatrix}$$

- Normalize:

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

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

# Solution Part 1: Normalized gradients

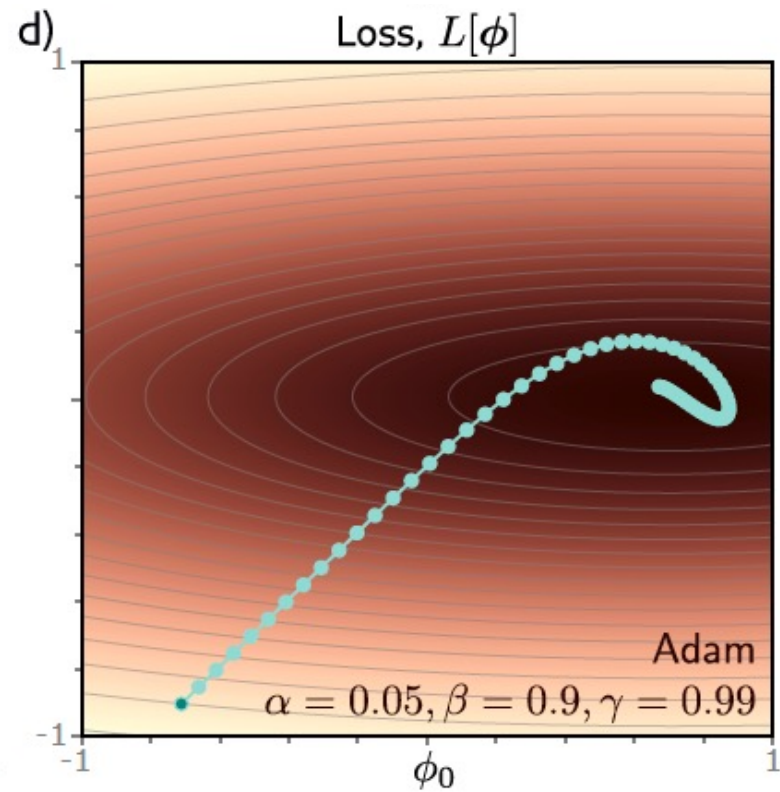
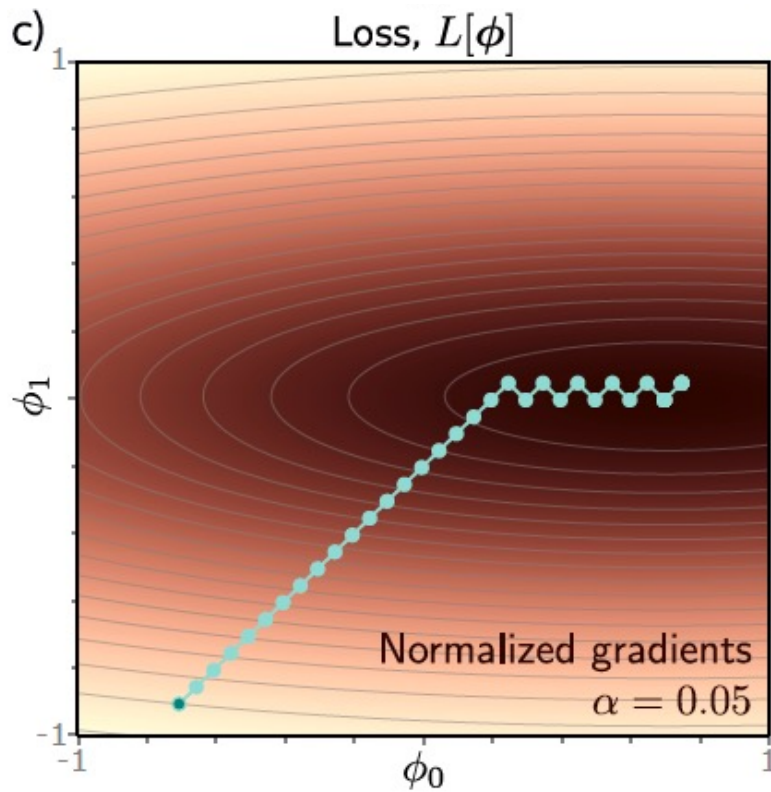


- algorithm moves downhill a fixed distance  $\alpha$  along each coordinate
- makes good progress in both directions
- but will not converge unless it happens to land exactly at the minimum

# 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  $\left\{ \begin{array}{l} \tilde{\mathbf{m}}_{t+1} \leftarrow \frac{\mathbf{m}_{t+1}}{1 - \beta^{t+1}} \quad \mathbf{m}_{t=0} = 0 \\ \tilde{\mathbf{v}}_{t+1} \leftarrow \frac{\mathbf{v}_{t+1}}{1 - \gamma^{t+1}} \quad \mathbf{v}_{t=0} = 0 \end{array} \right.$
- 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 – can be fixed, on a schedule or loss dependent
- Momentum Parameters

In class poll: <https://piazza.com/class/m5v834h9pcatx/post/32>

# 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

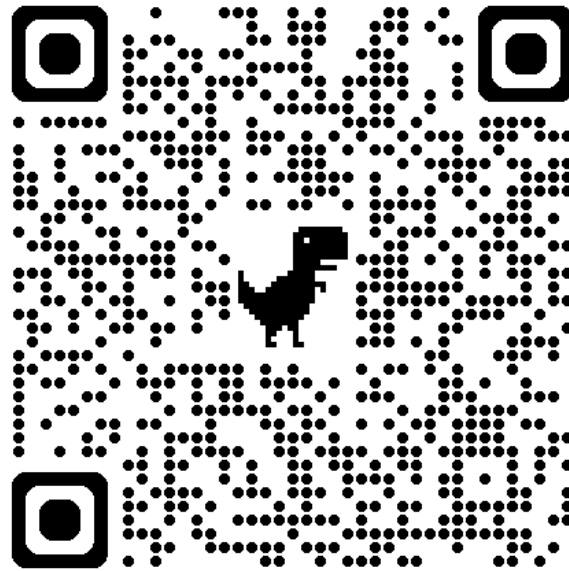
# Next

- Gradient of Deep Networks: Chain Rule, backpropagation and automated (scalable) gradient calculations
- Initialization
- Measuring training performance and how to improve
- Network regularization

----- End of Foundational Concepts -----

- CNNs
- Residual Networks
- Transformers

Feedback?



[Link](#)