BOSTON UNIVERSITY

Deep Learning for Data Science
DS 542

Lecture 06
Fitting Models

Slides originally by Thomas Gardos.

Images from [Understanding Deep Learning] unless otherwise cited.

Announcements

- SCC tutorial at today’s discussion section
  - Don’t miss it!
  - 3:35PM @ CAS 313

Recap: Loss Functions

- Originally the functions we minimize when training our models
    - Ad-hoc preferences on error tradeoffs if we cannot fit perfectly
- Last lecture
    - Least squared error as a real loss function for lost profits in industry
    - Maximum likelihood estimation to derive loss functions based on problem structure and modeling assumptions

Recap: Maximum Likelihood Estimation

1. Think of models as predicting probability distributions, not point estimates.
2. Pick model parameters maximizing the likelihood of the observed data.
   a. $\hat{\phi}$ = argmax$_\phi$ Π_i Pr(y_i | f[x,ϕ])
   b. $\hat{\phi}$ = argmax$_\phi$ log (Π_i Pr(y_i | f[x,ϕ])) log is motonic, avoids underflow
   c. $\hat{\phi}$ = argmax$_\phi$ Σ_i log Pr(y_i | f[x,ϕ])) log distributes over product
   d. **$\hat{\phi}$ = argmin$_\phi$ Σ_i -log Pr(y_i | f[x,ϕ]))** switch to minimizing by flipping signs
   e. L[ϕ] ⁇ Σ_i -log Pr(y_i | f[x,ϕ]) maybe? but might simplify further…

^^ loss function is what we minimize, but often can simplify more based on problem and choices of probability distributions

Recap: Regressing Gaussian Distribution

1. Model outputs a normal distribution parameterized by μ and σ.

$$\mu = f[x, \phi]$$

$$Pr(y_i, x_i) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left[-\frac{(y_i - f[x_i, \phi])^2}{2\sigma^2}\right]$$

2. Maximum likelihood estimation equivalent to least squares.

$$\hat{\phi} = \text{argmin}_{\phi} \sum_i -\log\left[\frac{1}{\sqrt{2\pi\sigma^2}}\exp\left[-\frac{(y_i - f[x_i, \phi])^2}{2\sigma^2}\right]\right]$$

$$\hat{\phi} = \text{argmin}_{\phi} \sum_i -\log\left[\exp\left[-\frac{(y_i - f[x_i, \phi])^2}{2\sigma^2}\right]\right]$$

$$\hat{\phi} = \text{argmin}_{\phi} \sum_i -\left[-\frac{(y_i - f[x_i, \phi])^2}{2\sigma^2}\right]$$

$$\hat{\phi} = \text{argmin}_{\phi} \sum_i (y_i - f[x_i, \phi])^2$$

$$L[x, \phi] = \sum_i (y_i - f[x_i, \phi])^2$$

Least squares!

Recap: Regressing Binary/Multiclass Classification

Two issues intertwined

- Derivation of loss function
- Finagling arbitrary neural network output into probability distributions.
  - This part is somewhat arbitrary, but these ways tend to work…
  - Both equations would simplify to a cross-entropy formula without that finagling

$$\hat{\phi} = \text{argmin}_{\phi} \sum_{i} -\log f_{y_i}[x_i, \phi]$$

Recap: Regressing a Binary Output

1. Model outputs a Bernoulli distribution parameterized by λ after using sigmoid to get probabilities from f[].

$$y \in \{0,1\}$$

$$\lambda = \text{sig}[f[x,\phi]] \in [0,1]$$

$$Pr(y=1\,|\,\lambda) = \lambda$$

$$Pr(y=0\,|\,\lambda) = 1-\lambda$$

$$Pr(y\,|\,\lambda) = (1-\lambda)^{1-y}\lambda^{y}$$

2. Maximum likelihood estimation equivalent to binary cross-entropy loss.

$$\hat{\phi} = \text{argmin}_{\phi} \sum_i -\log\left[(1-\text{sig}[f[x,\phi]])^{1-y}\ \text{sig}[f[x,\phi]]^{y}\right]$$

$$\hat{\phi} = \text{argmin}_{\phi} \sum_i \left[-(1-y)\log(1-\text{sig}[f[x,\phi]]) - y\log\text{sig}[f[x,\phi]]\right]$$

$$\hat{\phi} = \text{argmin}_{\phi} \sum_i \left[-(y=0)\log(Pr_f(y=0\,|\,x,\phi])) - (y=1)\log(Pr_f(y=1\,|\,x,\phi))\right]$$

$$\hat{\phi} = \text{argmin}_{\phi} \sum_i -\log(Pr_f(y=y_i\,|\,x_i,\phi))$$

Cross-entropy losses penalize with the negative log of the modeled probability, so low predicted probabilities give high losses.

Recap: Regressing Multiple Classes

Model has one output per class and uses softmax to map to get clean probabilities.

$$\text{softmax}_k(\mathbf{z}) = \frac{\exp[z_k]}{\sum_{k'} \exp[z_{k'}]}$$

$$Pr(y = k \,|\, \mathbf{f}[x, \phi]) = \text{softmax}_k(\mathbf{f}[x, \phi])$$

2. Maximum likelihood estimation equivalent to multiclass cross-entropy loss.

$$\hat{\phi} = \text{argmin}_\phi \sum_i -\log\left[\text{softmax}_{y_i}(\mathbf{f}[x, \phi])\right]$$

$$\hat{\phi} = \text{argmin}_\phi \sum_i -\log\left[\frac{\exp[f_{y_i}[x, \phi]]}{\sum_{k'} \exp[f_{k'}[x, \phi]}\right]$$

$$\hat{\phi} = \text{argmin}_\phi \sum_i \left[-\log \exp[f_{y_i}[x, \phi]] + \log \sum_{k'} \exp[f_{k'}[x, \phi]\right]$$

$$\hat{\phi} = \text{argmin}_\phi \sum_i \left[-f_{y_i}[x, \phi] + \log \sum_{k'} \exp[f_{k'}[x, \phi]\right]$$

Biggest term? Negative log of numerator

Fitting models

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

Code Preview

The Magic Code

```
# use Adam optimizer with low learning rate.
optimizer = torch.optim.Adam(parameters, lr=1e-4)

for i in range(epochs):
    # reset gradient tracking
    optimizer.zero_grad(set_to_none=True)

    # run the current network against all training inputs at once.
    # so this is a whole batch version of gradient descent.
    prediction = run_network(train_X)

    # compute average squared error loss.
    loss = 0.5 * torch.mean((train_y - prediction) ** 2)
    # backpropagation of the loss. to be covered in lecture 7.
    loss.backward()

    # update parameters. to be covered today.
    optimizer.step()
```

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:

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

or for short:

$$L\left[\phi\right]$$

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

Training

- Loss function:

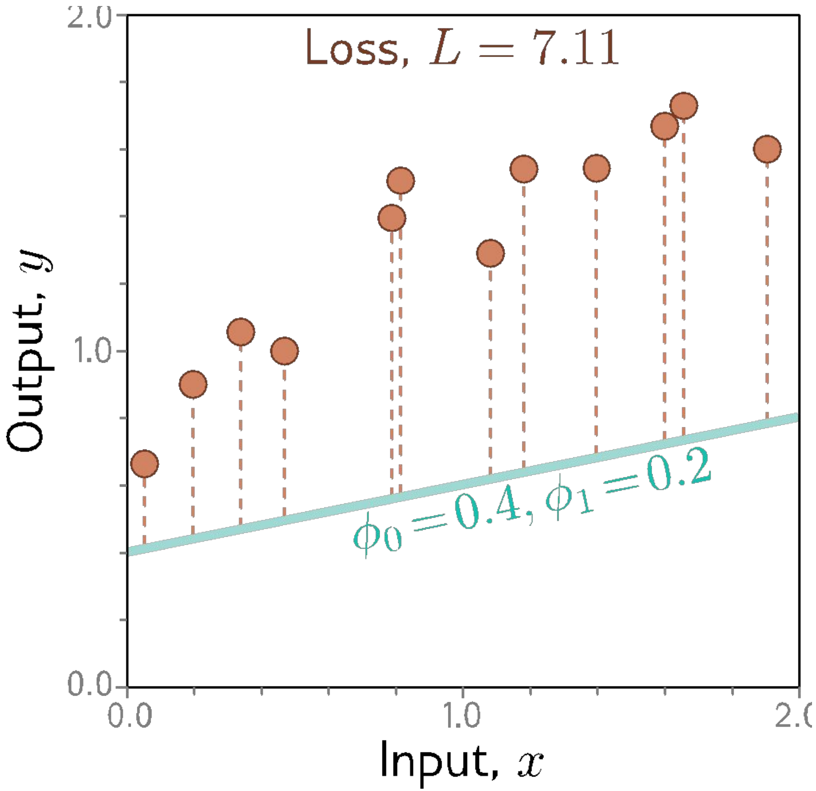
$$L[\phi]$$

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

- Find the parameters that minimize the loss:

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

Example: 1D Linear regression loss function



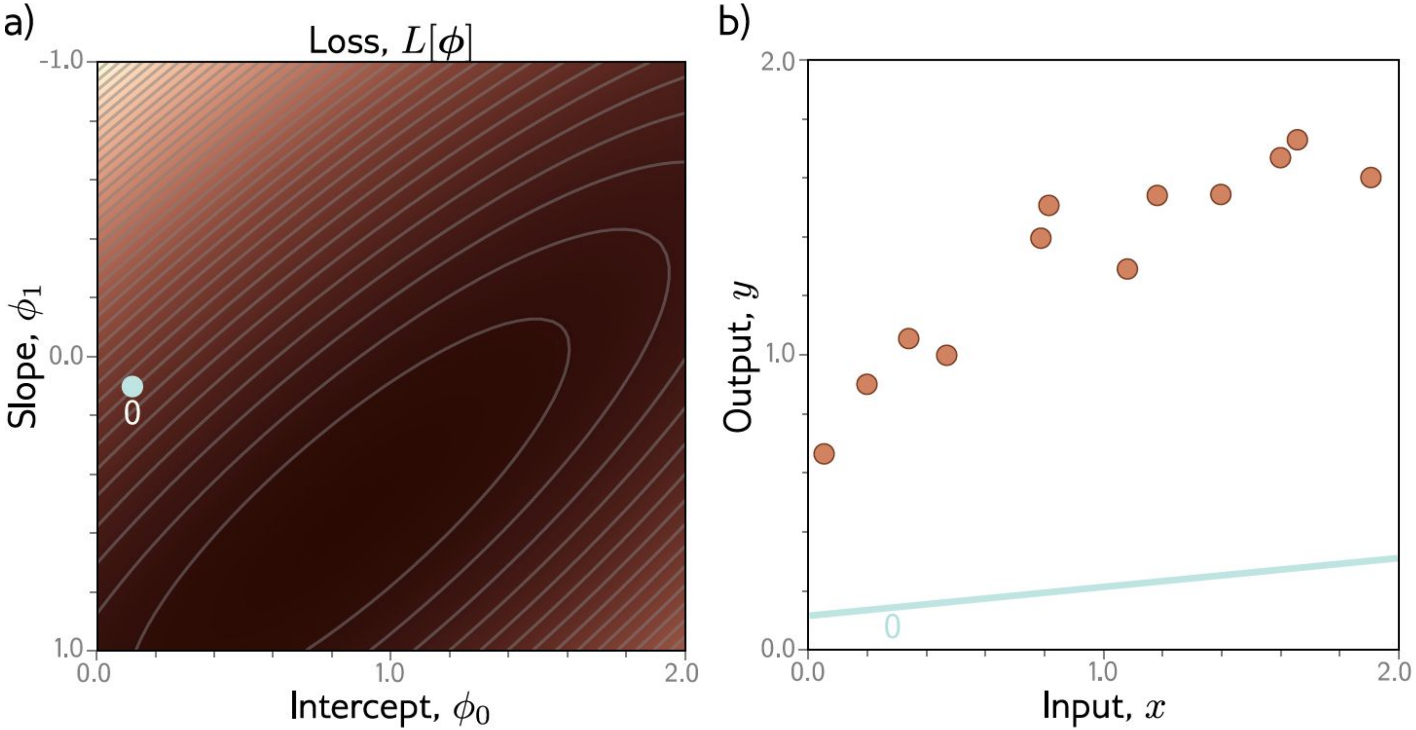
Loss function:

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

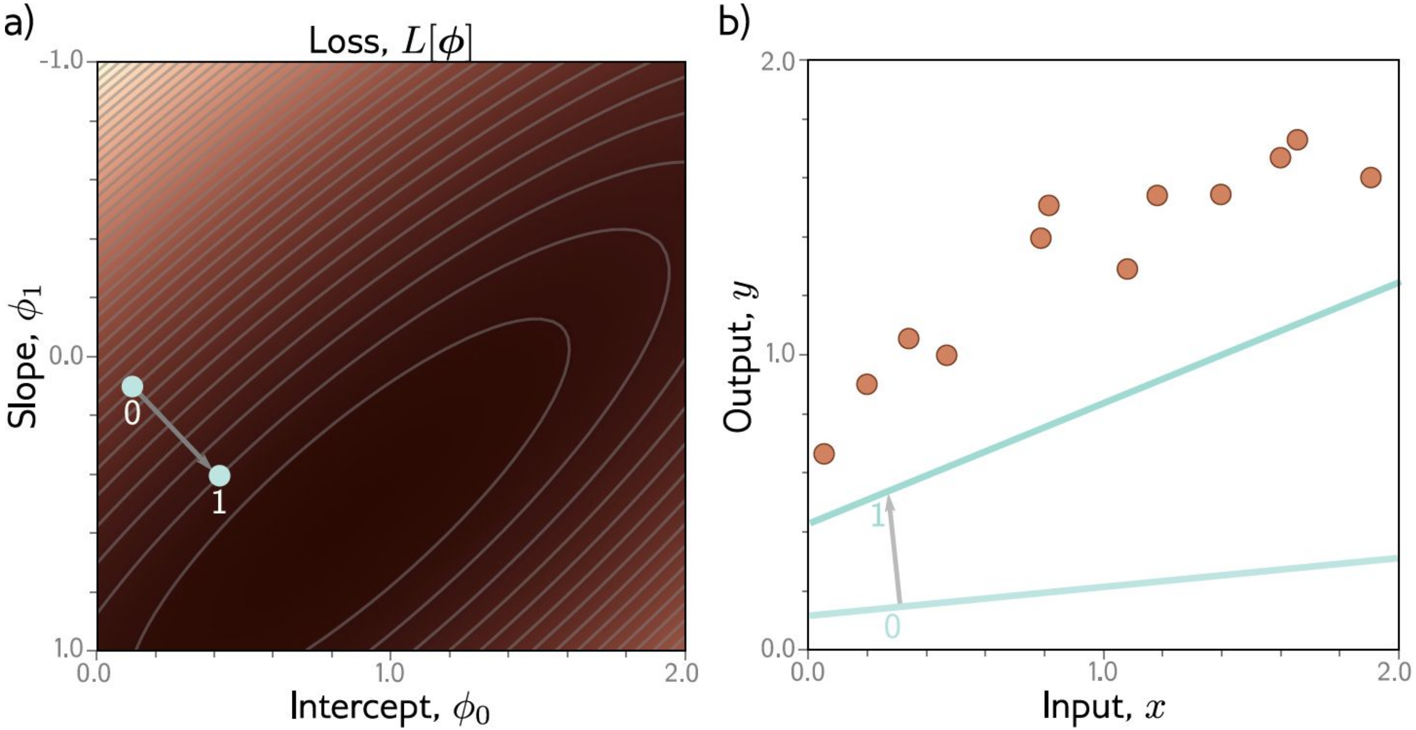
$$= \sum_{i=1}^{I} (\phi_0 + \phi_1 x_i - y_i)^2$$

"Least squares loss function"

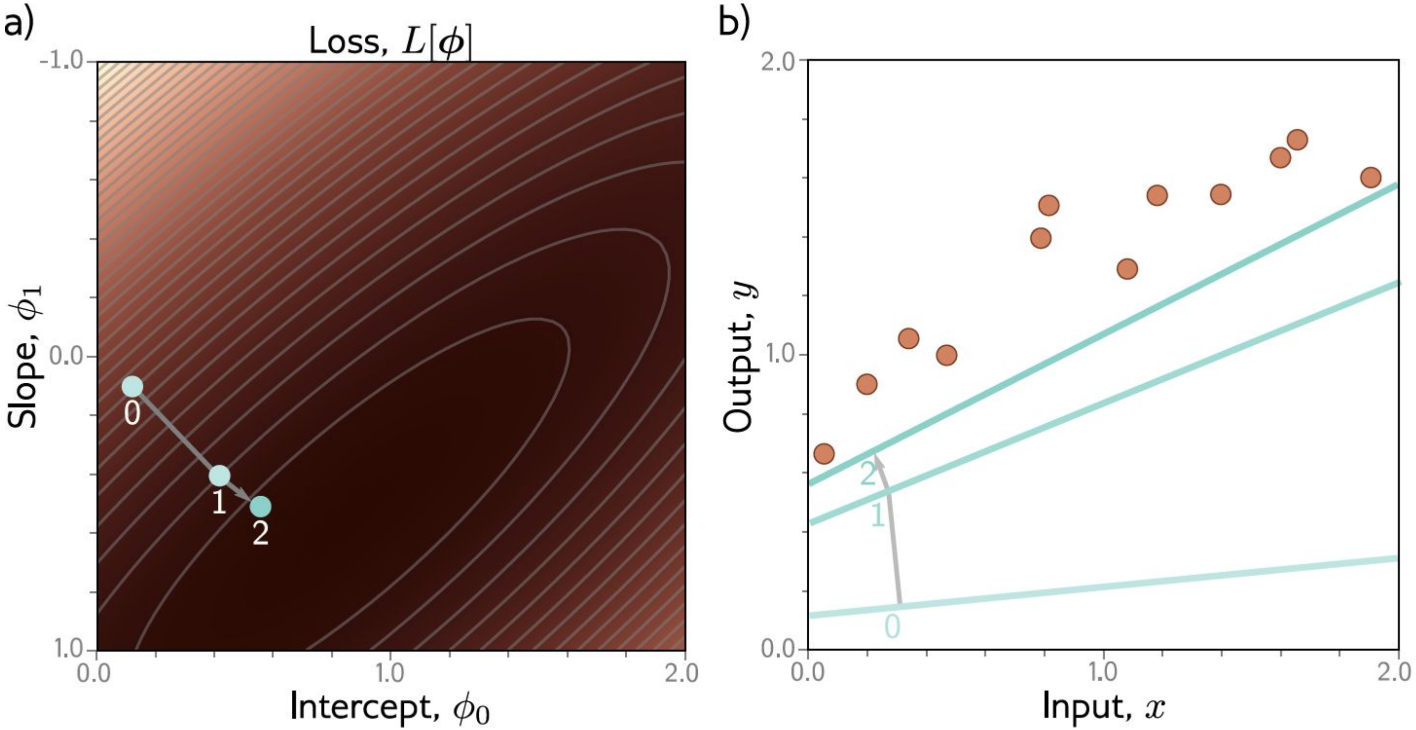
Example: 1D Linear regression training



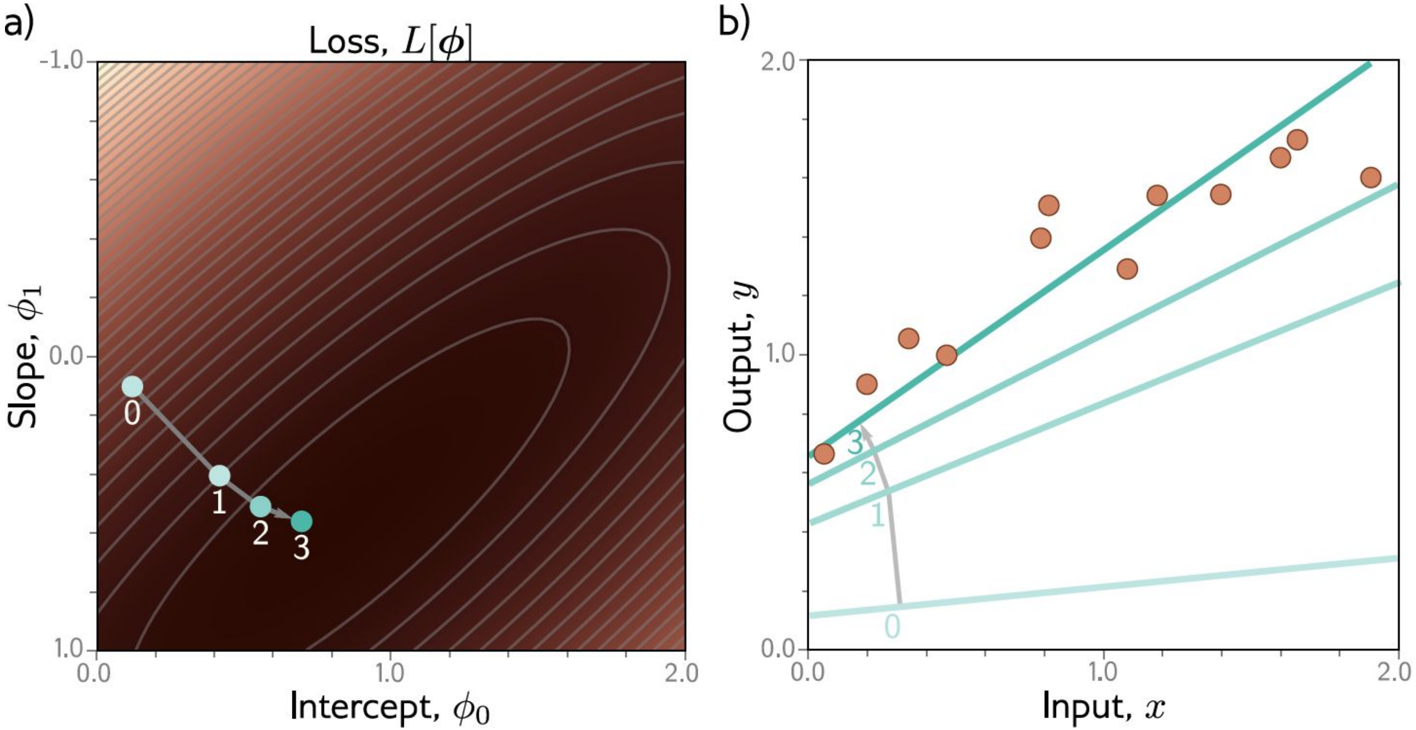
Example: 1D Linear regression training



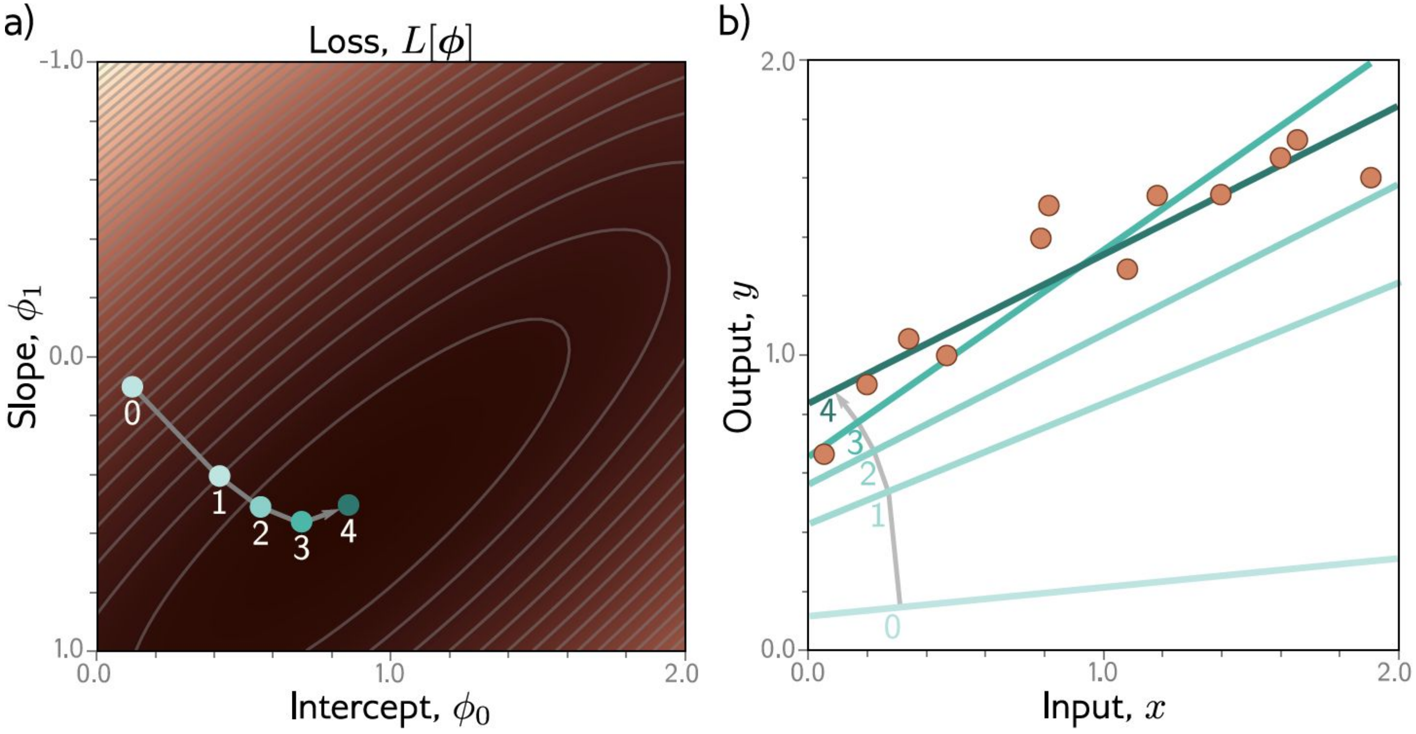
Example: 1D Linear regression training



Example: 1D Linear regression training



Example: 1D Linear regression training



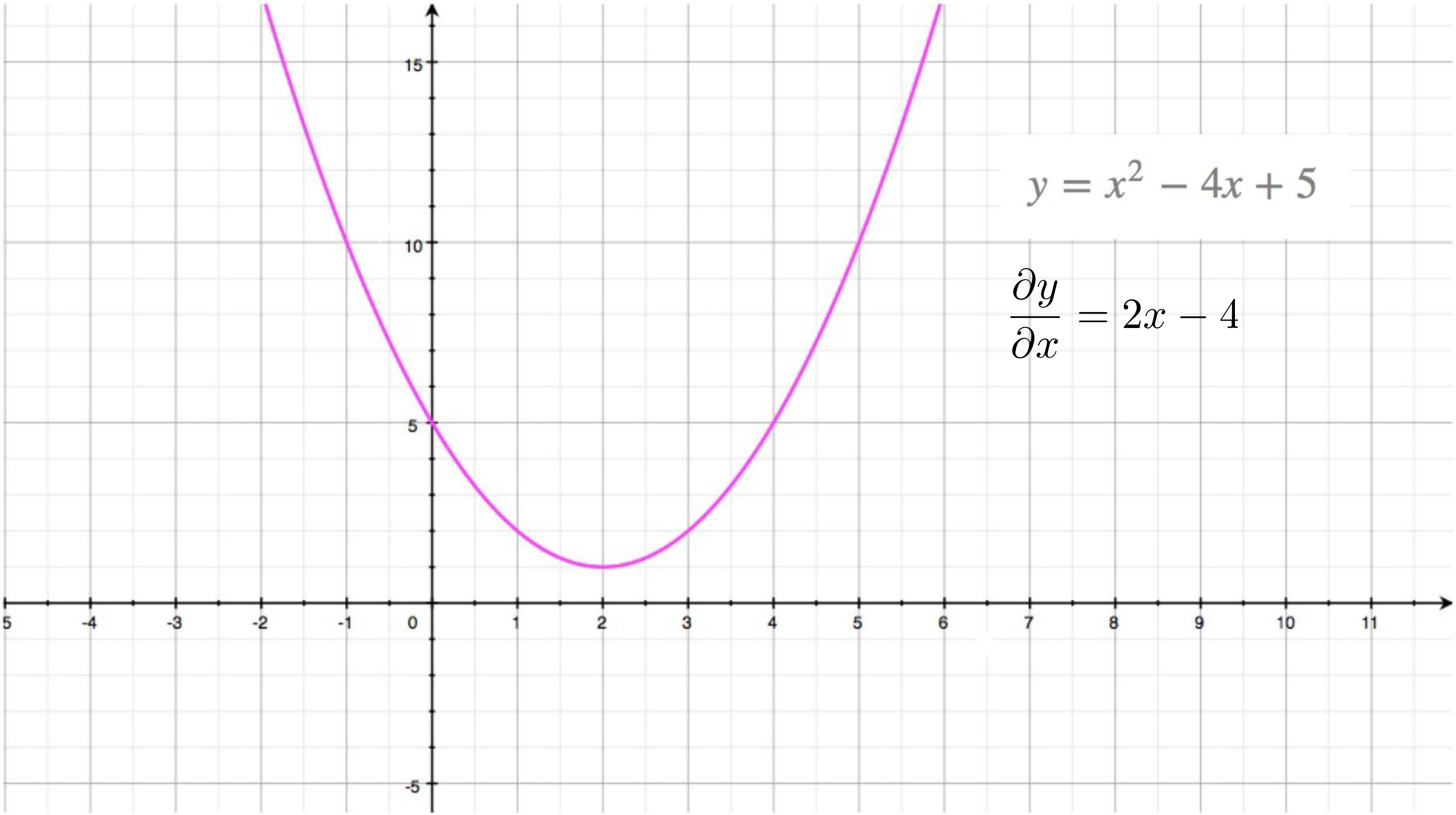
This technique is known as gradient descent

Fitting models

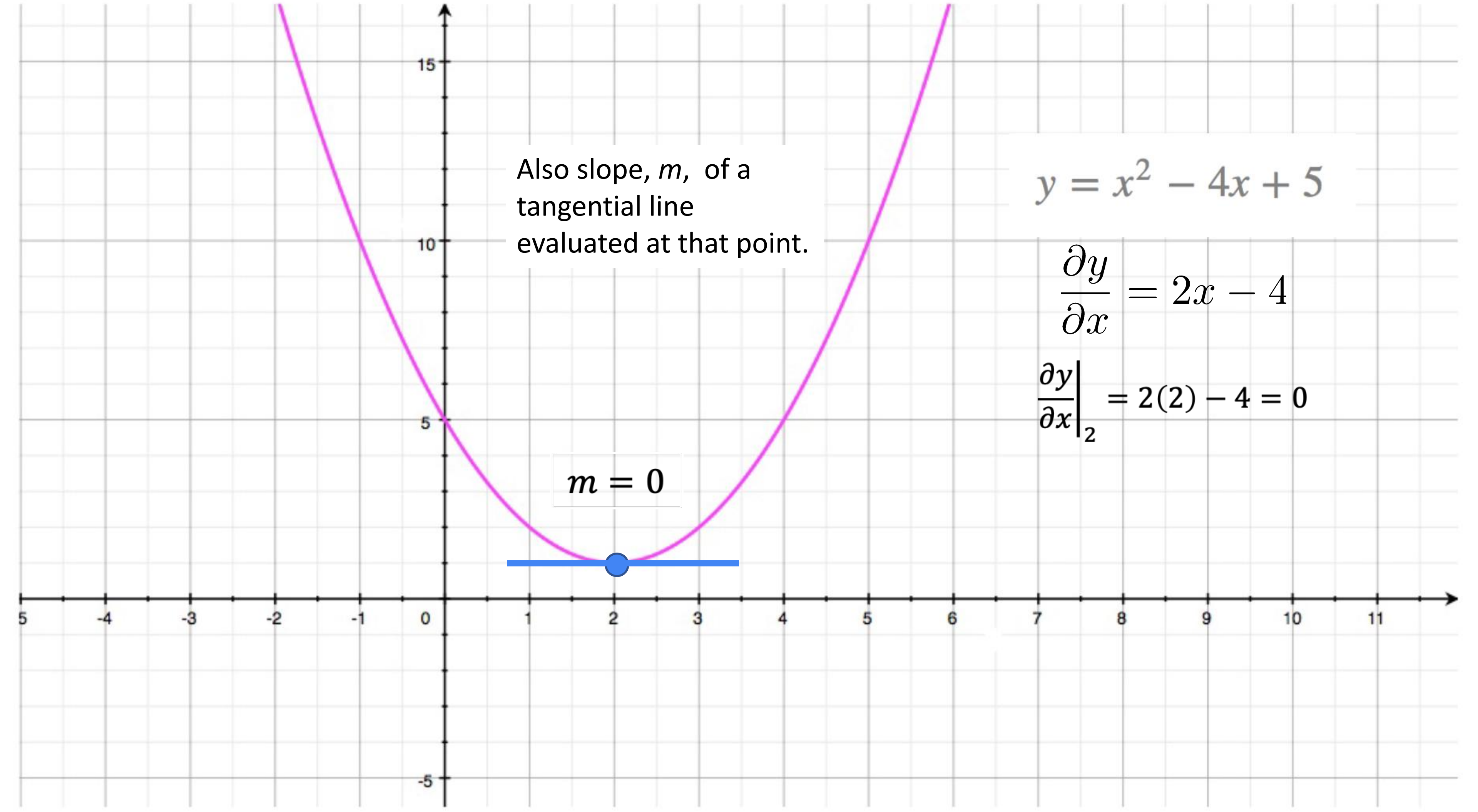
- Code Preview
- Math overview
- Gradient descent algorithm
  - Linear regression example
  - Gabor model example
- Stochastic gradient descent
- Momentum
- Adam

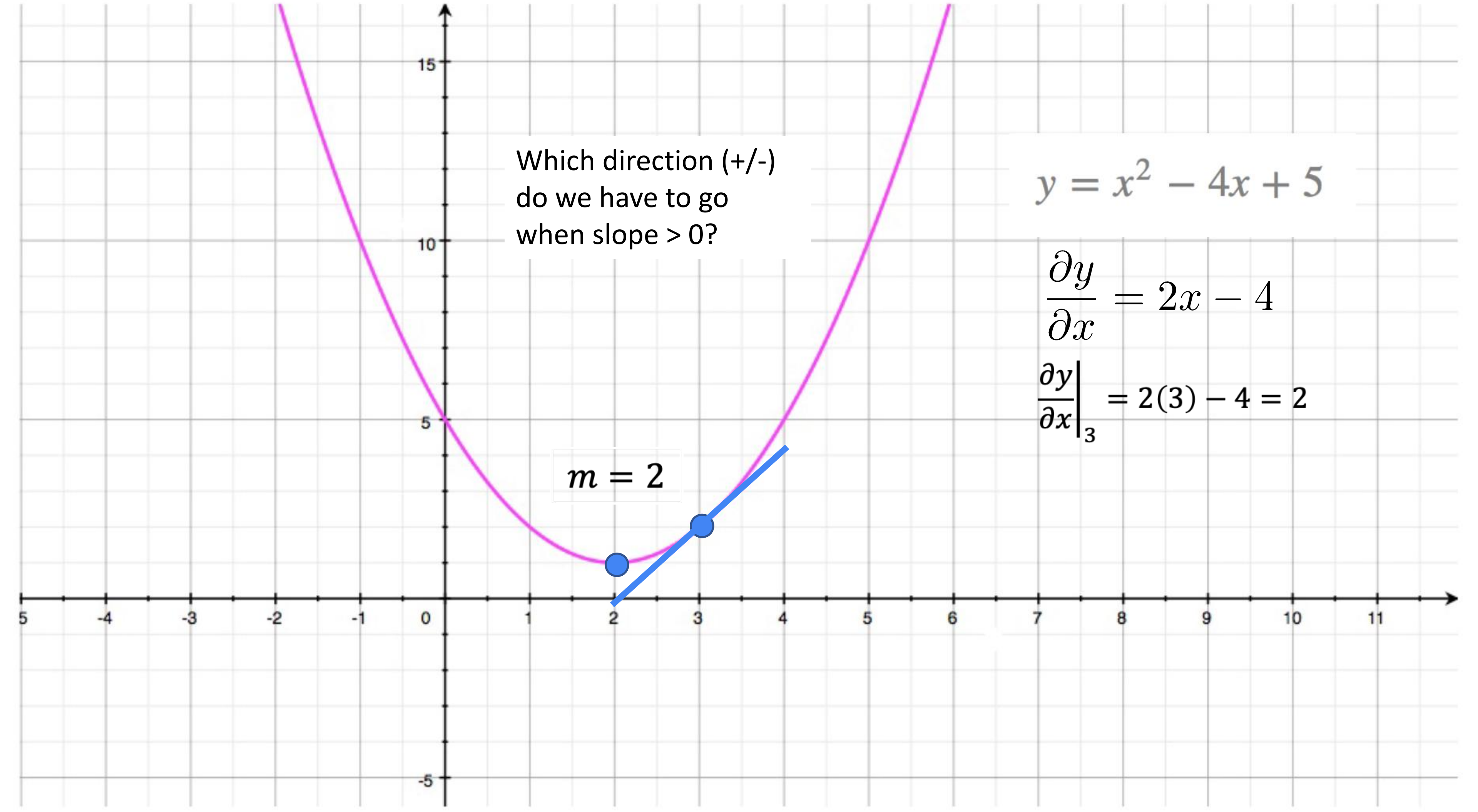
Definitions

- 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 $\lim_{h \to 0} \frac{f(a+h)-f(a)}{h}$ exists.
  - You can approximate the derivative with this limit.
- gradient
  - the degree and direction of steepness of a graph at any point



$y = x^2 - 4x + 5$
$\frac{\partial y}{\partial x} = 2x - 4$
15
10
5
0
-5
5
-4
-3
-2
-1
1
2
3
4
5
6
7
8
9
10
11


Also slope, *m*, of a tangential line evaluated at that point.
$y = x^2 - 4x + 5$
$\frac{\partial y}{\partial x} = 2x - 4$
$\left.\frac{\partial y}{\partial x}\right|_2 = 2(2) - 4 = 0$
$m = 0$
15
10
5
-5
5
-4
-3
-2
-1
0
1
2
3
4
5
6
7
8
9
10
11

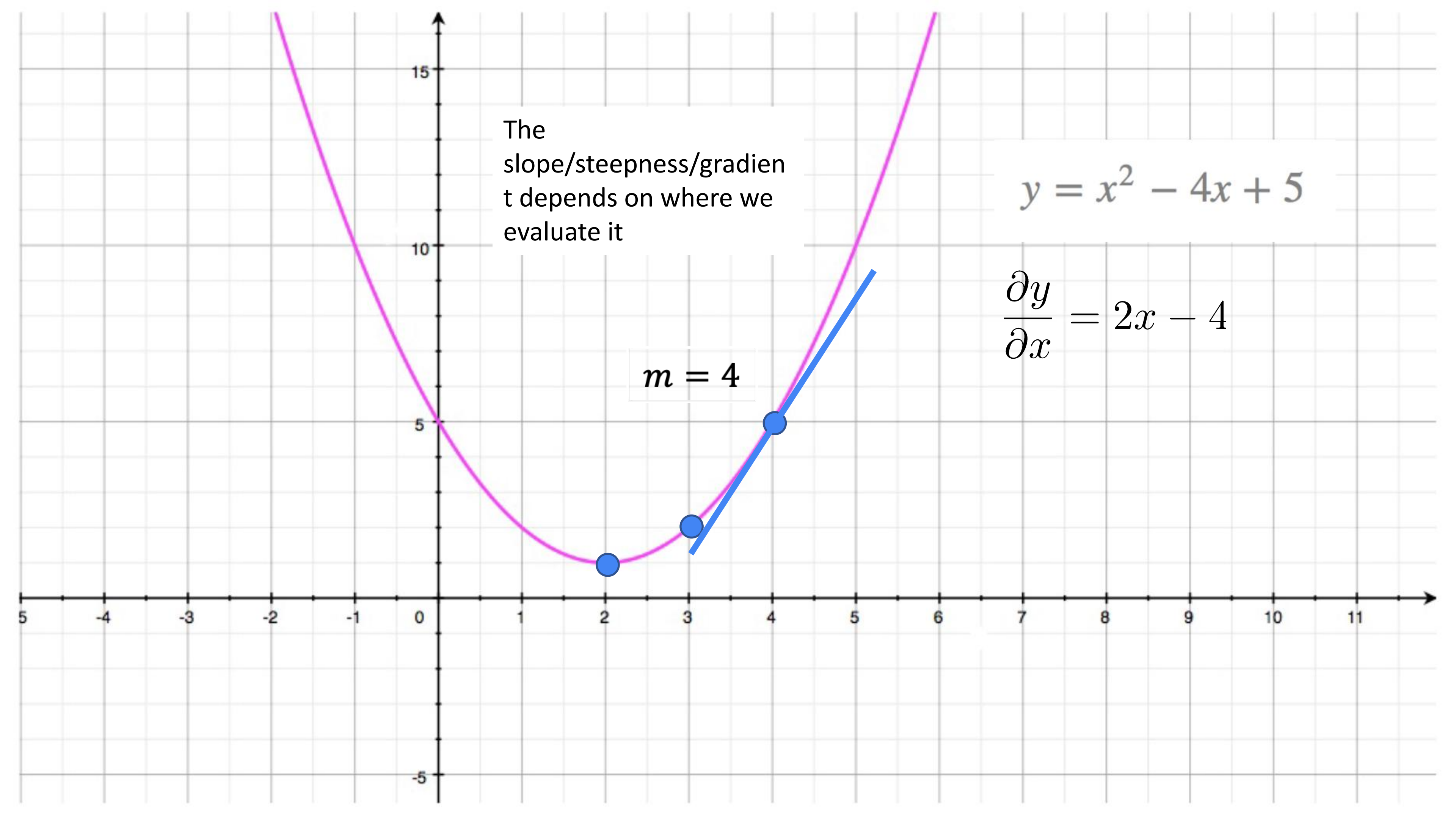


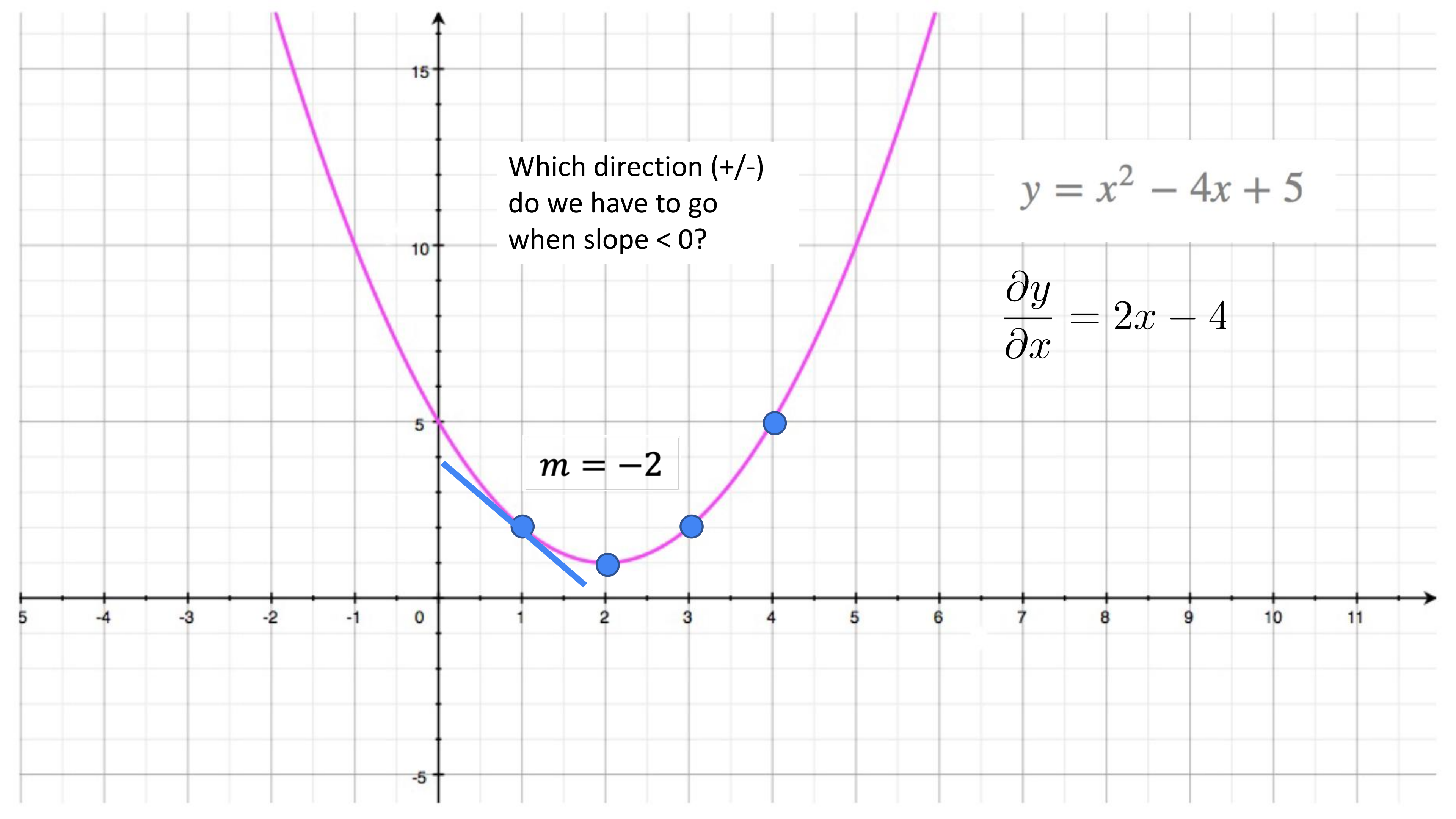
Which direction (+/-) do we have to go when slope > 0?

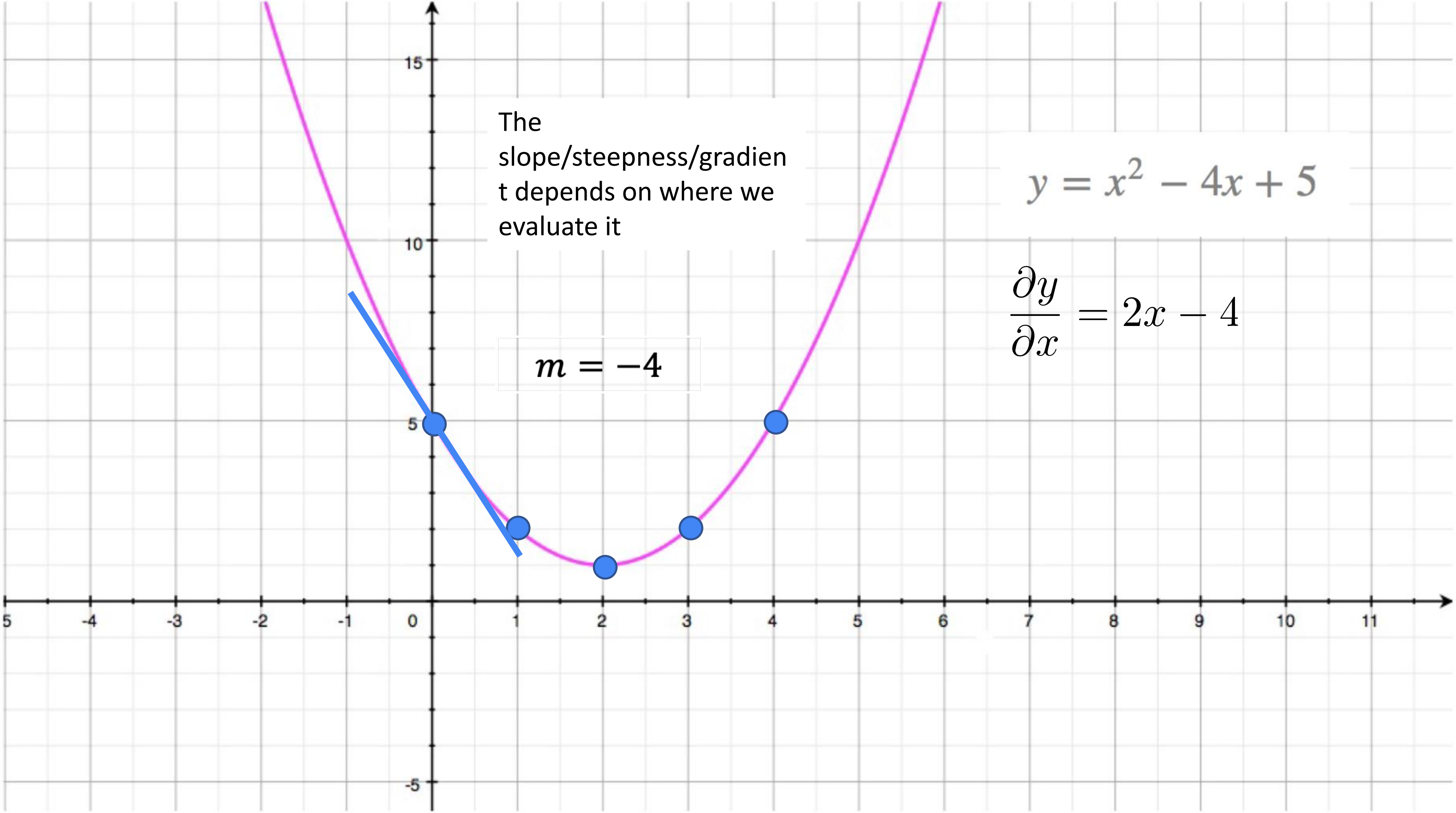
$$y = x^2 - 4x + 5$$

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

$$\left.\frac{\partial y}{\partial x}\right|_3 = 2(3) - 4 = 2$$


The slope/steepness/gradien t depends on where we evaluate it
$y = x^2 - 4x + 5$
$\frac{\partial y}{\partial x} = 2x - 4$
$m = 4$
15
10
5
-5
5
-4
-3
-2
-1
0
1
2
3
4
5
6
7
8
9
10
11


Which direction (+/-)
do we have to go
when slope < 0?
$y = x^2 - 4x + 5$
$\frac{\partial y}{\partial x} = 2x - 4$
$m = -2$
15
10
5
0
-5
-4
-3
-2
-1
1
2
3
4
5
6
7
8
9
10
11



The slope/steepness/gradient depends on where we evaluate it

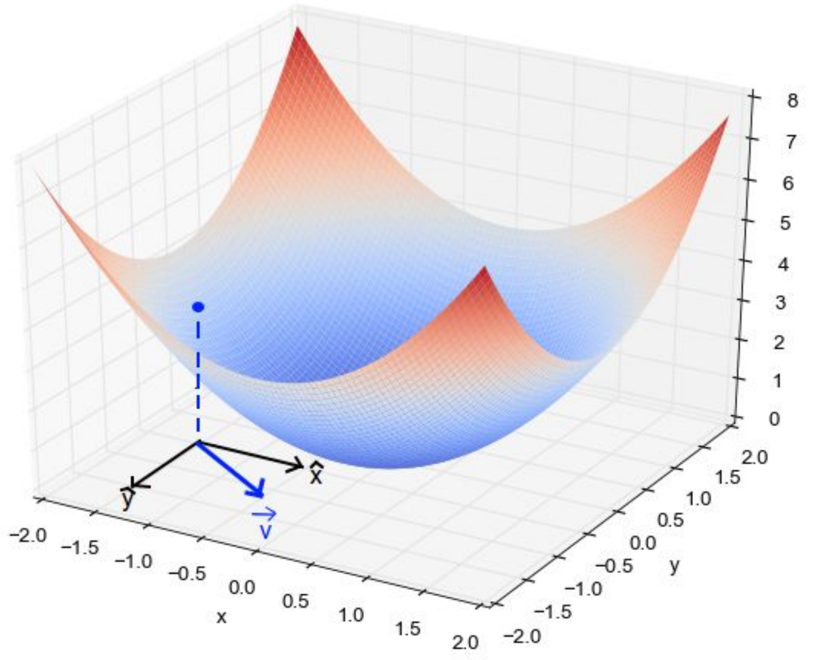
$$y = x^2 - 4x + 5$$

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

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

- Code Preview
- 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:

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

Also notated as $\nabla_w L$

Step 2. Update the parameters according to the rule:

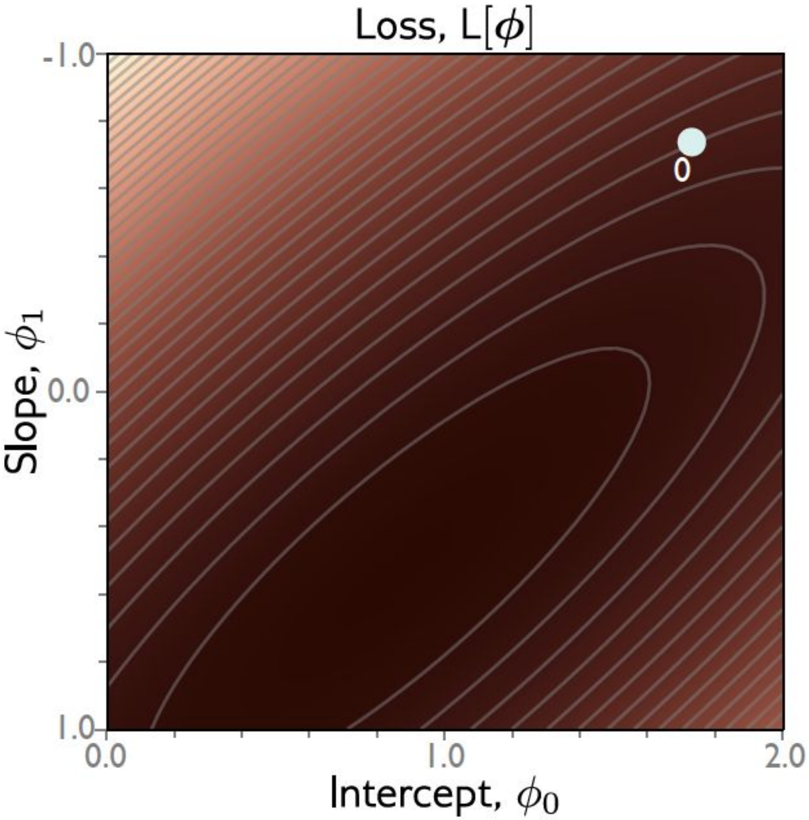
$$\phi \longleftarrow \phi - \alpha \frac{\partial L}{\partial \phi},$$

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

Fitting models

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

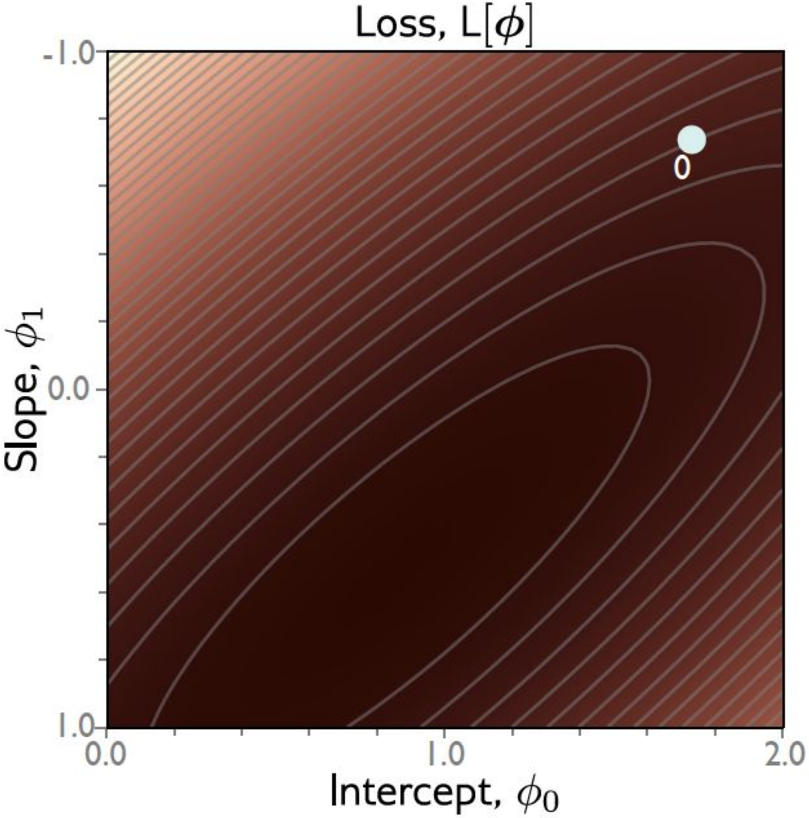
Gradient descent



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

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

Gradient descent

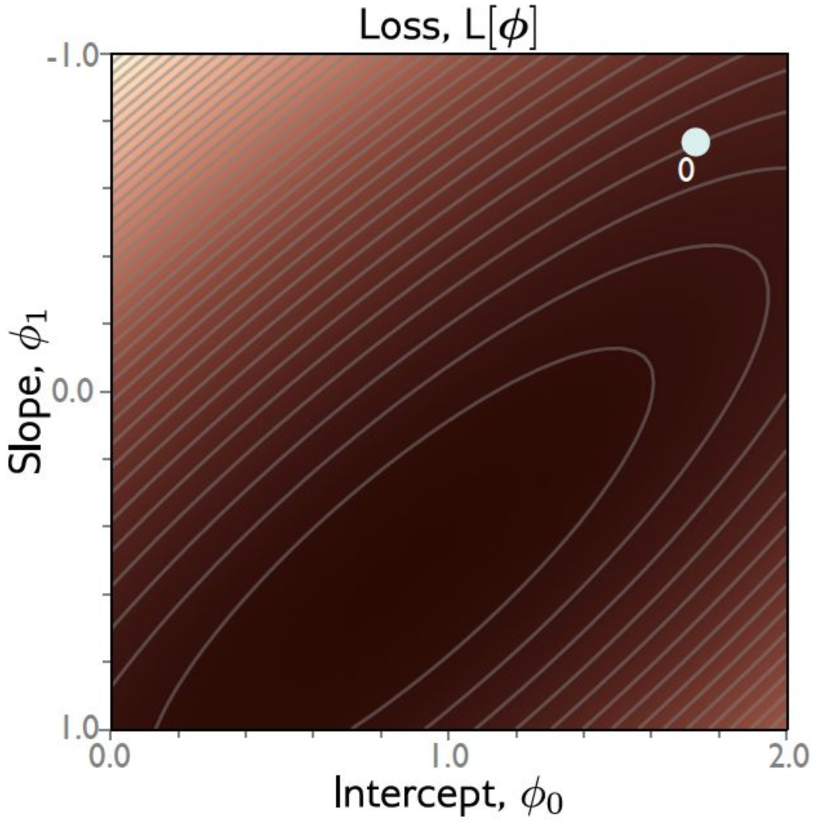


Step 1: Compute derivatives (slopes of function) with
Respect to the parameters

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

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

Gradient descent



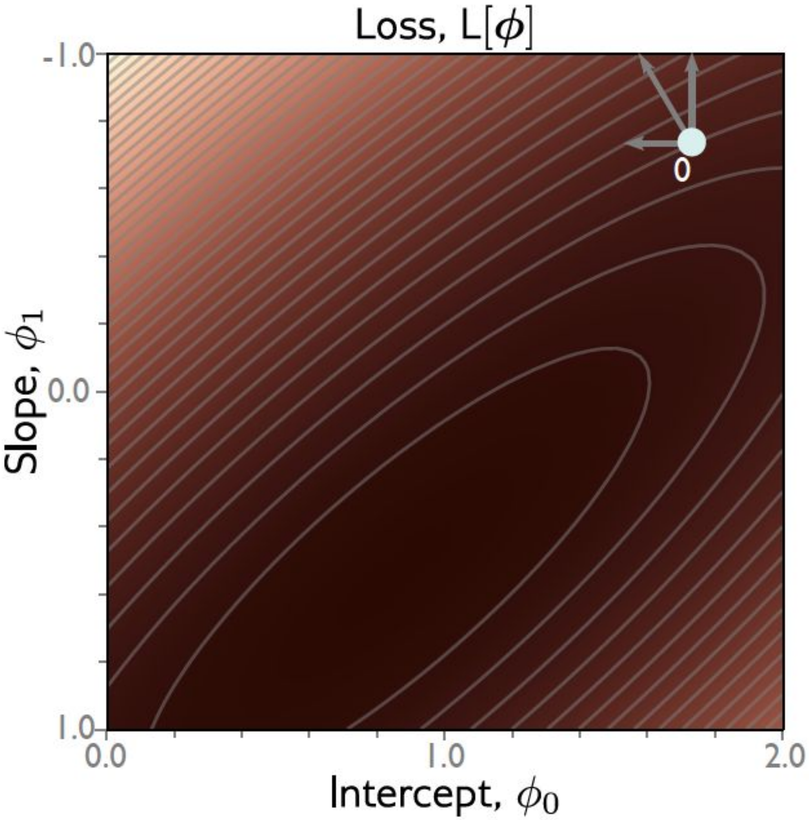
Step 1: Compute derivatives (slopes of function) with
Respect to the parameters

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

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

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

Gradient descent

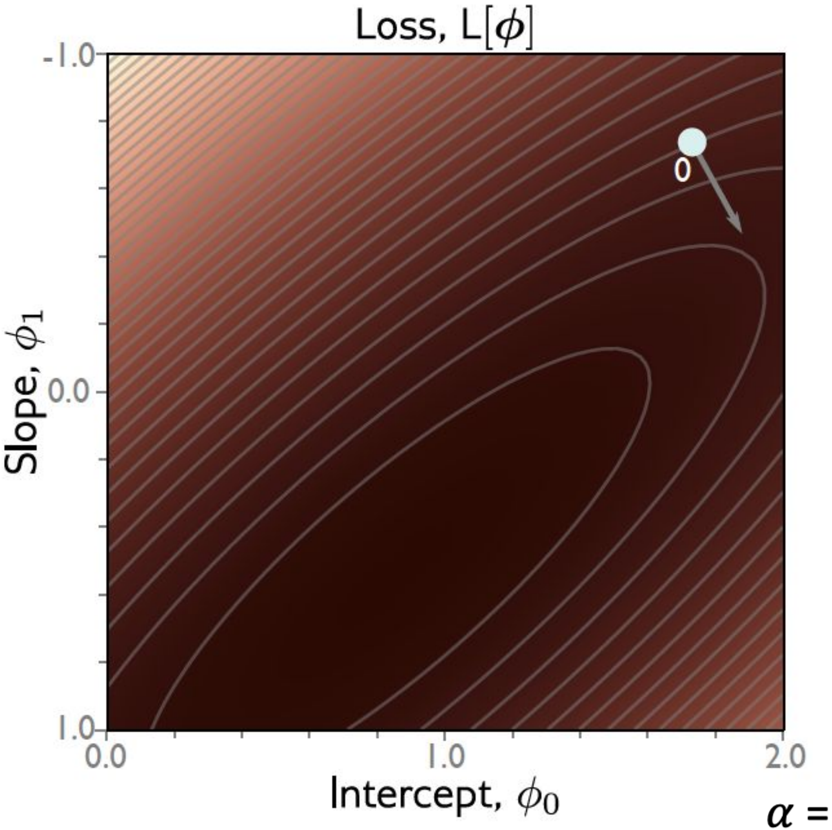


Step 1: Compute derivatives (slopes of function) with
Respect to the parameters

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

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

Gradient descent



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

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

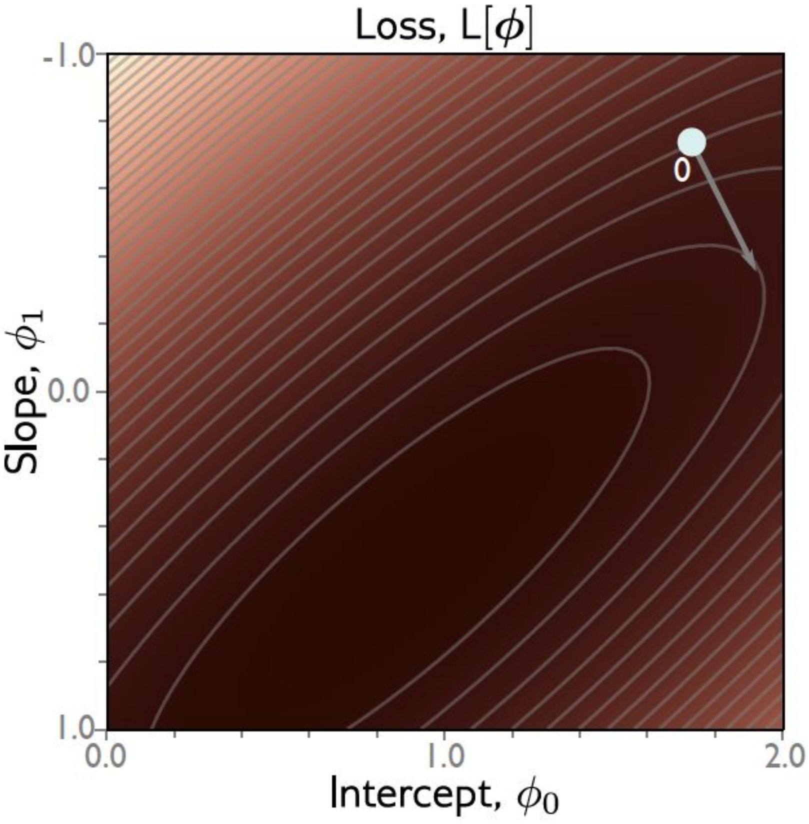
$$\frac{\partial \ell_i}{\partial \phi} = \begin{bmatrix} \frac{\partial \ell_i}{\partial \phi_0} \\ \frac{\partial \ell_i}{\partial \phi_1} \end{bmatrix} = \begin{bmatrix} 2(\phi_0 + \phi_1 x_i - y_i) \\ 2x_i(\phi_0 + \phi_1 x_i - y_i) \end{bmatrix}$$

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



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

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

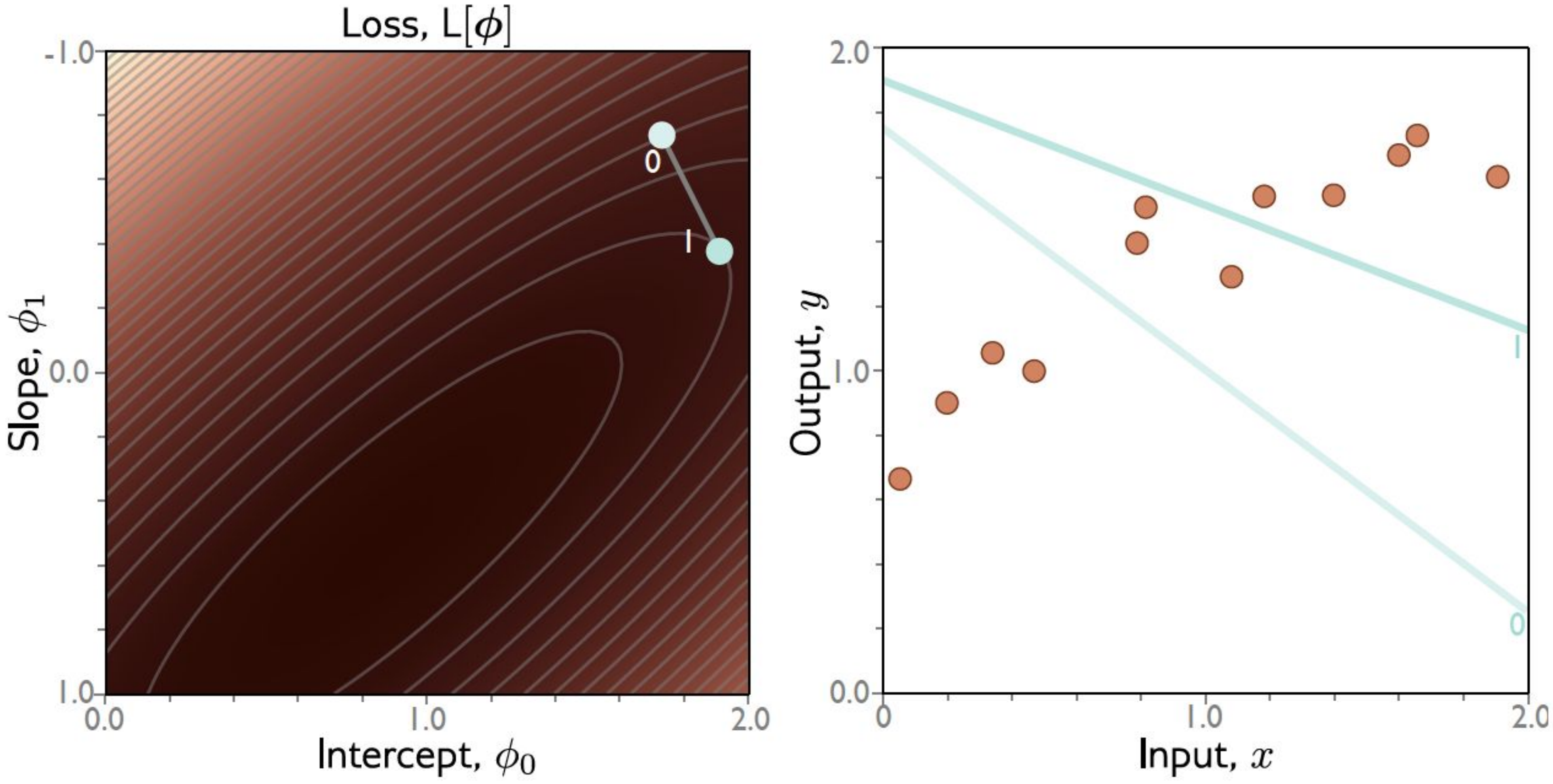
$$\frac{\partial \ell_i}{\partial \boldsymbol{\phi}} = \begin{bmatrix} \frac{\partial \ell_i}{\partial \phi_0} \\ \frac{\partial \ell_i}{\partial \phi_1} \end{bmatrix} = \begin{bmatrix} 2(\phi_0 + \phi_1 x_i - y_i) \\ 2x_i(\phi_0 + \phi_1 x_i - y_i) \end{bmatrix}$$

Step 2: Update parameters according to rule

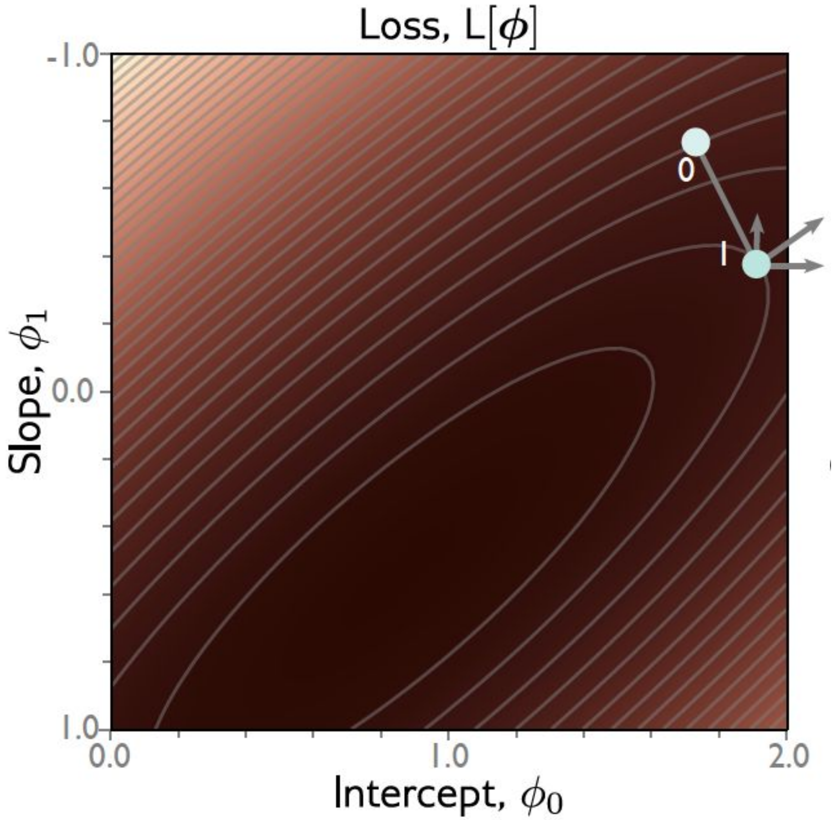
$$\boldsymbol{\phi} \longleftarrow \boldsymbol{\phi} - \alpha \frac{\partial L}{\partial \boldsymbol{\phi}}$$

α = step size

Gradient descent



Gradient descent



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

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

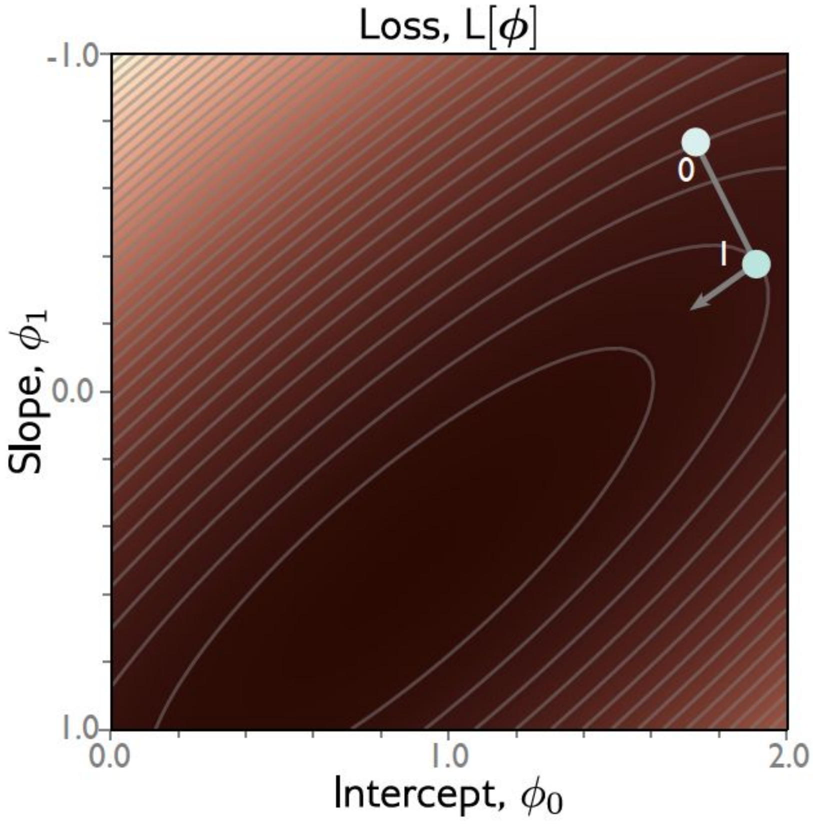
$$\frac{\partial \ell_i}{\partial \phi} = \begin{bmatrix} \frac{\partial \ell_i}{\partial \phi_0} \\ \frac{\partial \ell_i}{\partial \phi_1} \end{bmatrix} = \begin{bmatrix} 2(\phi_0 + \phi_1 x_i - y_i) \\ 2x_i(\phi_0 + \phi_1 x_i - y_i) \end{bmatrix}$$

Step 2: Update parameters according to rule

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

α = step size

Gradient descent



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

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

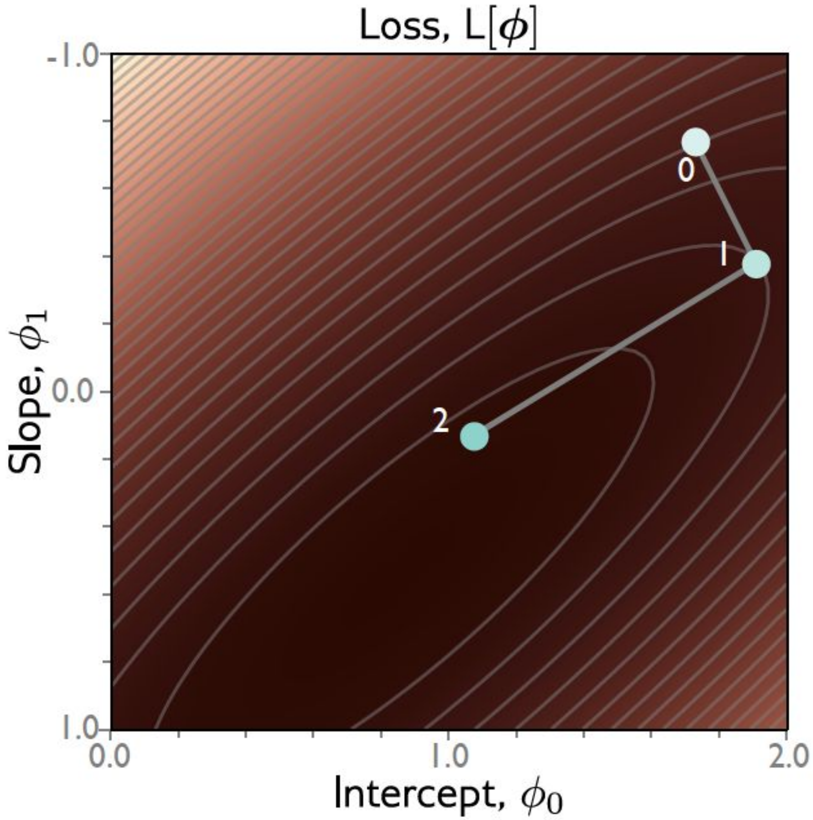
$$\frac{\partial \ell_i}{\partial \boldsymbol{\phi}} = \begin{bmatrix} \frac{\partial \ell_i}{\partial \phi_0} \\ \frac{\partial \ell_i}{\partial \phi_1} \end{bmatrix} = \begin{bmatrix} 2(\phi_0 + \phi_1 x_i - y_i) \\ 2x_i(\phi_0 + \phi_1 x_i - y_i) \end{bmatrix}$$

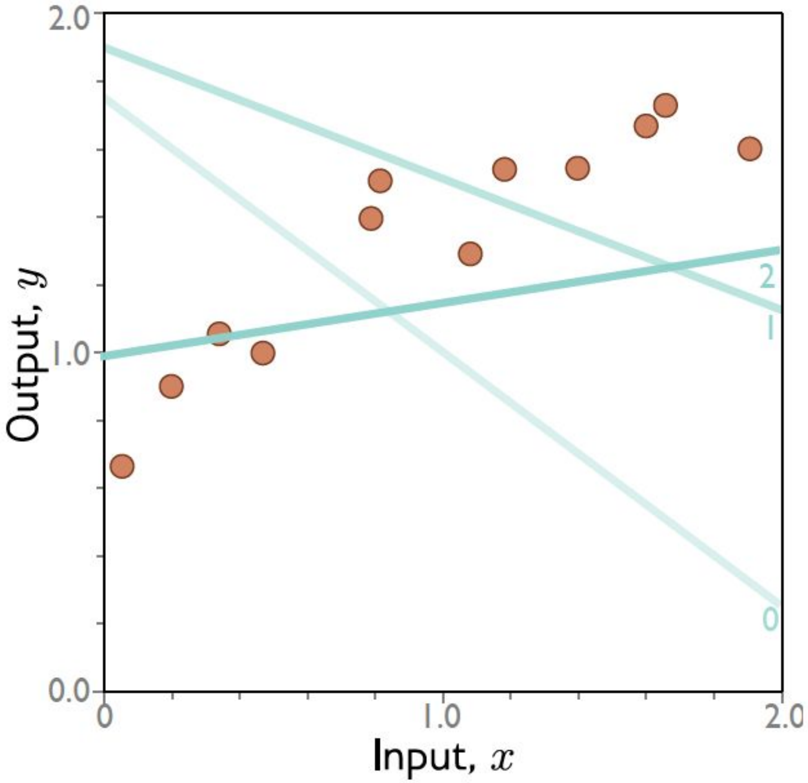
Step 2: Update parameters according to
rule

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

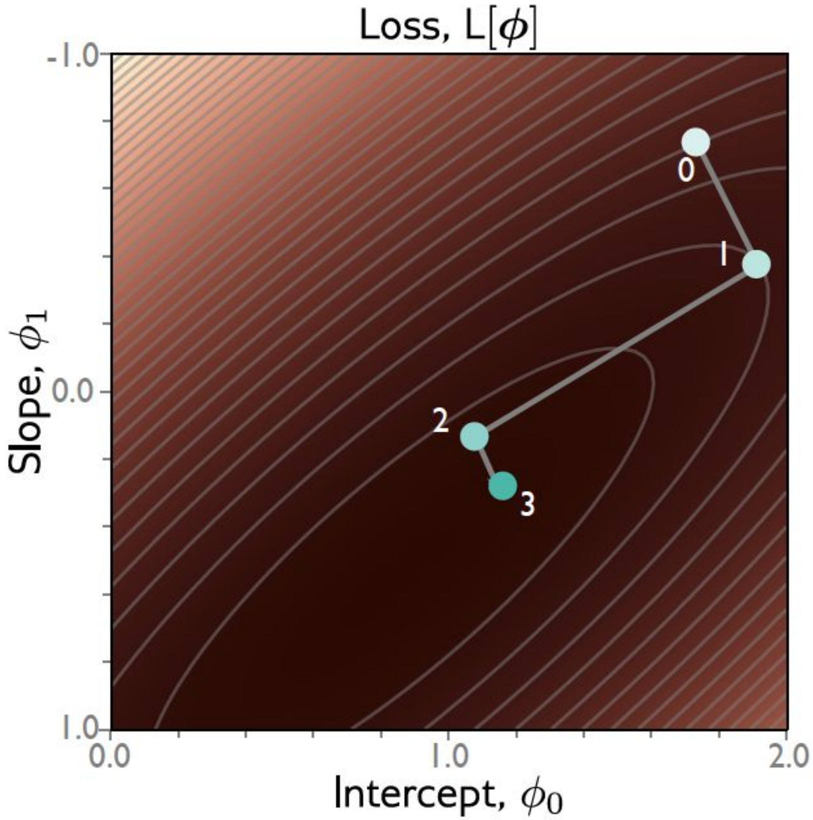
α = step size

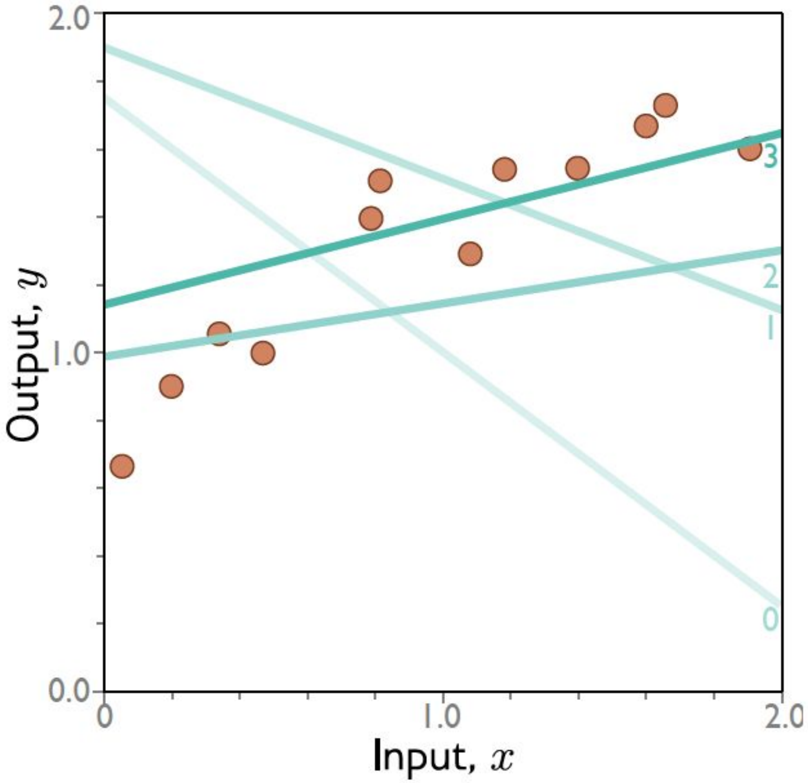
Gradient descent



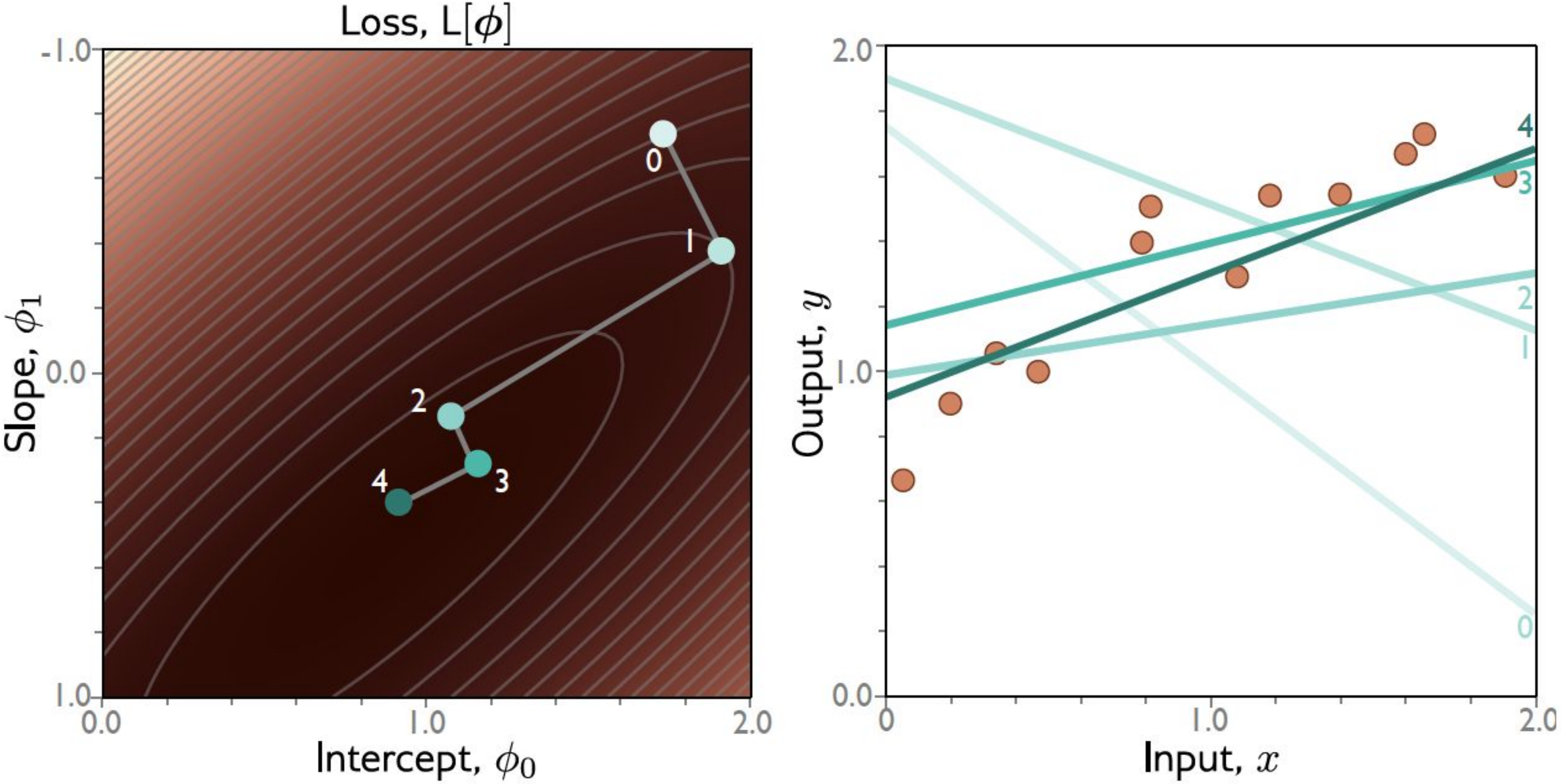


Gradient descent

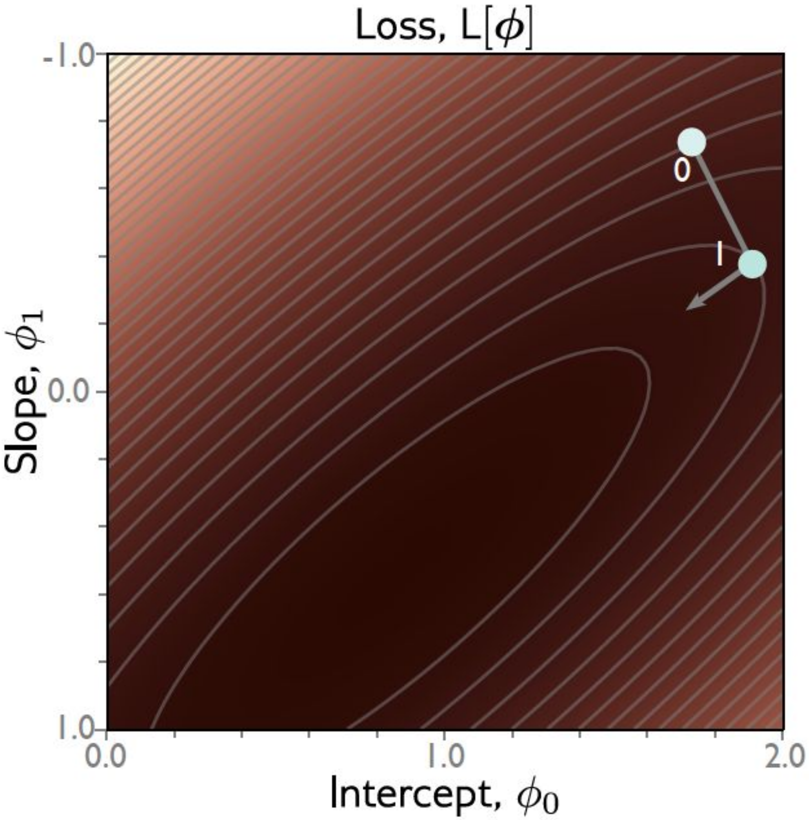




Gradient descent



Line Search

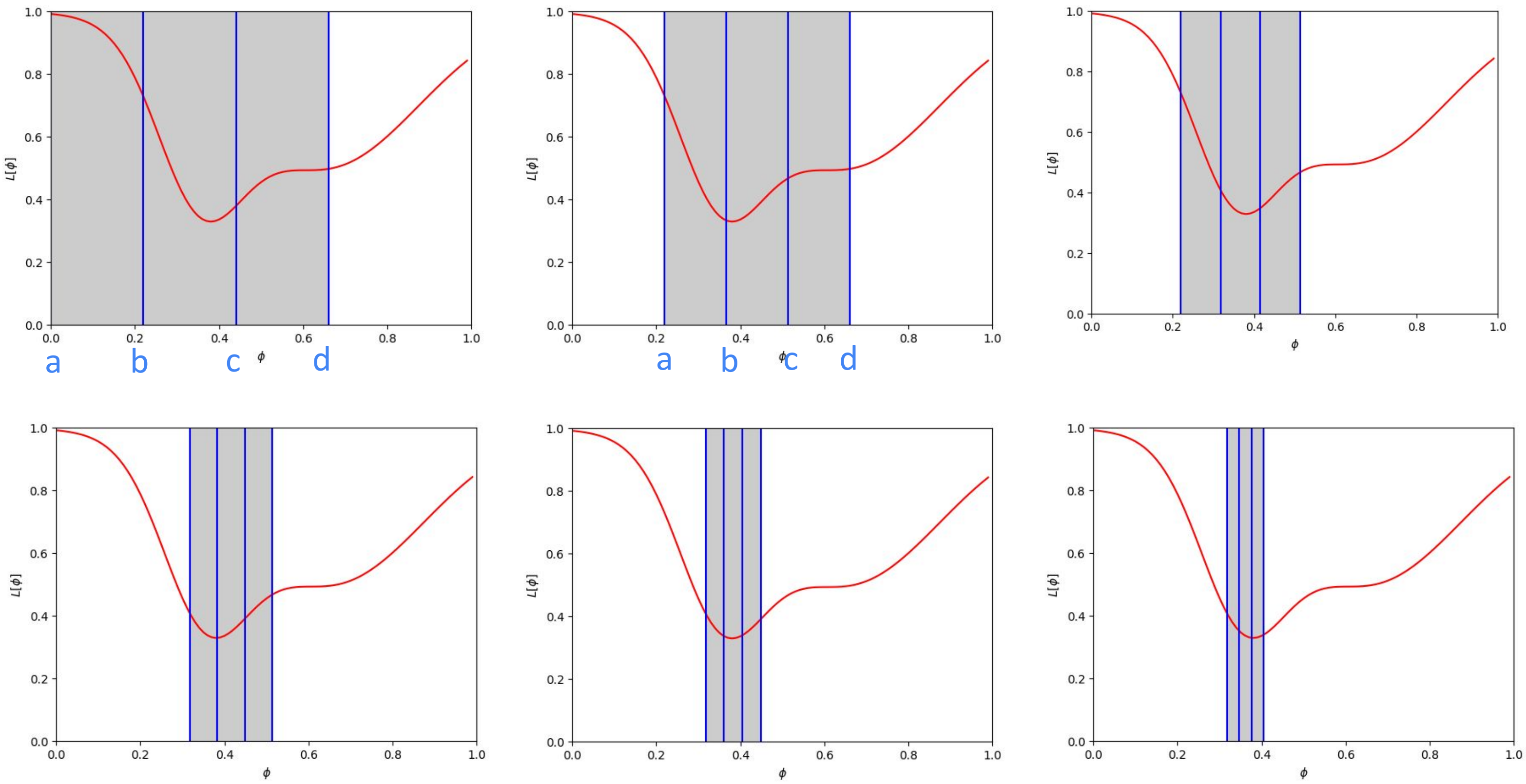


We can also search for the optimal *step size* at each iteration using *Line Search*

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

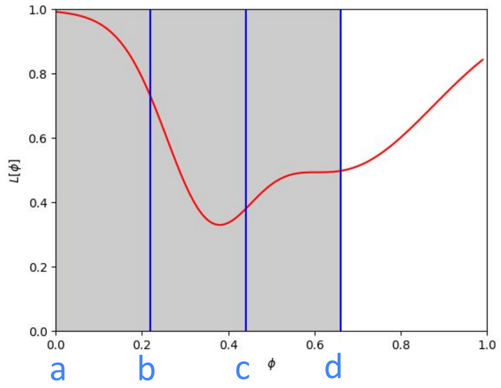
α = step size

Line Search (bracketing)



Line Search (bracketing)

- For each iteration you are evaluating loss four times
- Can be costly for more complex data types and loss calculations (e.g. image segmentation, ….)
- Not typically used ~~for computer vision~~ for large problems of any sort
    - But motivates heuristics changing learning rate during the training process.



Fitting models

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

Gabor Model

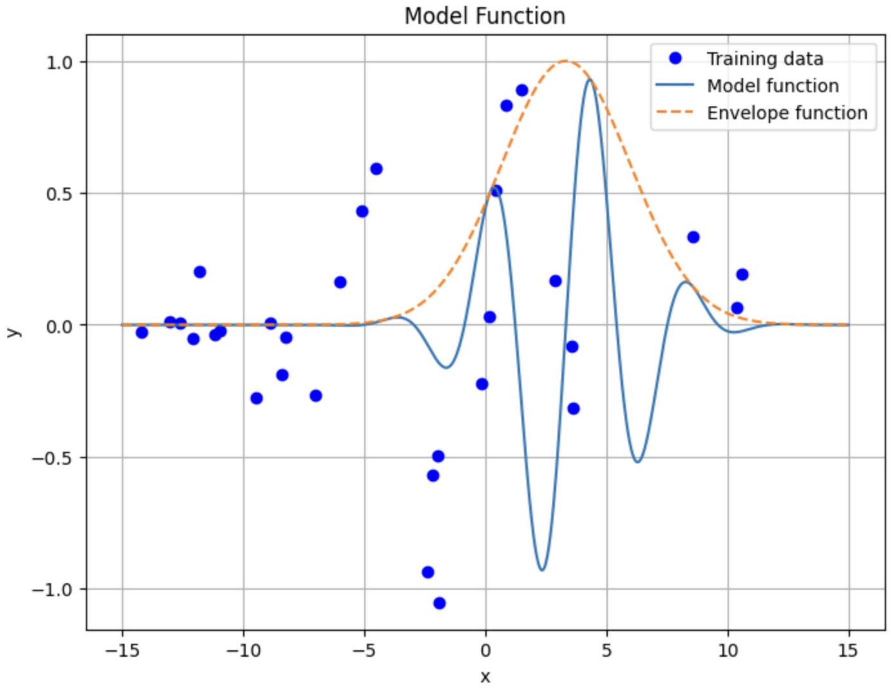
Linear model loss functions are always convex

Gabor modes are a more complex (non-convex) model that we can still visualize in 2D and 3D…

- Developed for image processing
- Looks for a signal of a particular frequency and alignment.
- Still differentiable, so we can reason about it similarly to linear models and neural networks.

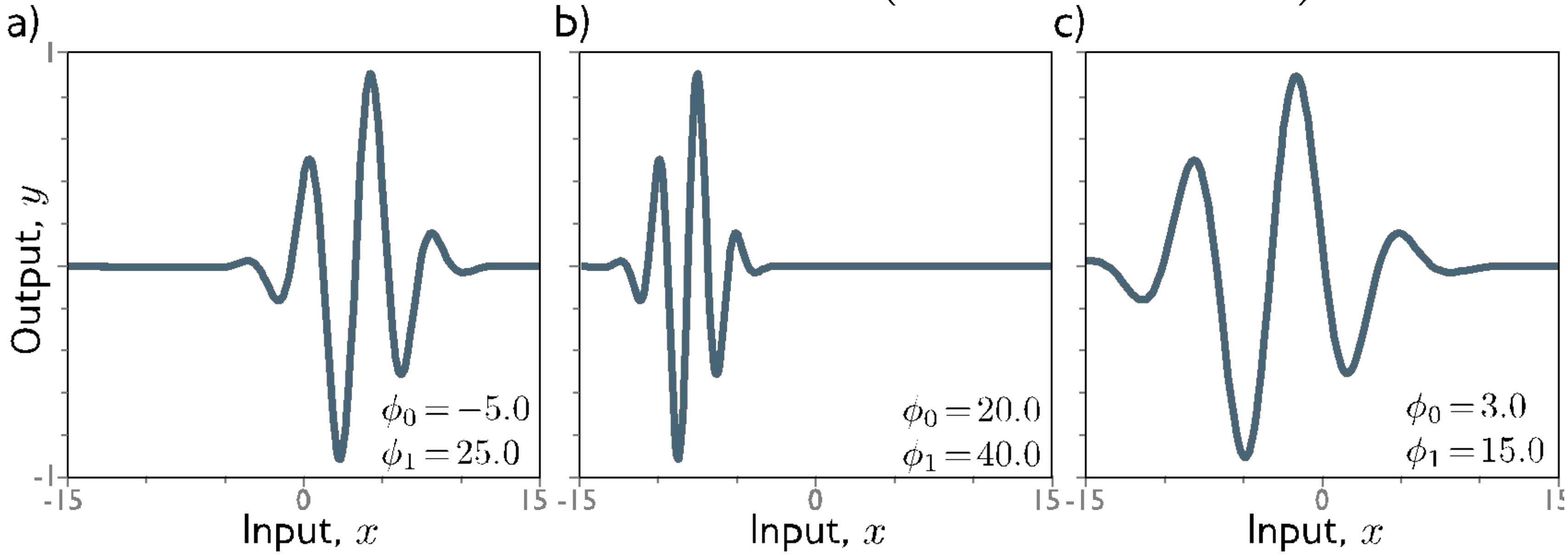
Gabor Model (with Envelope)

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



Gabor model

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

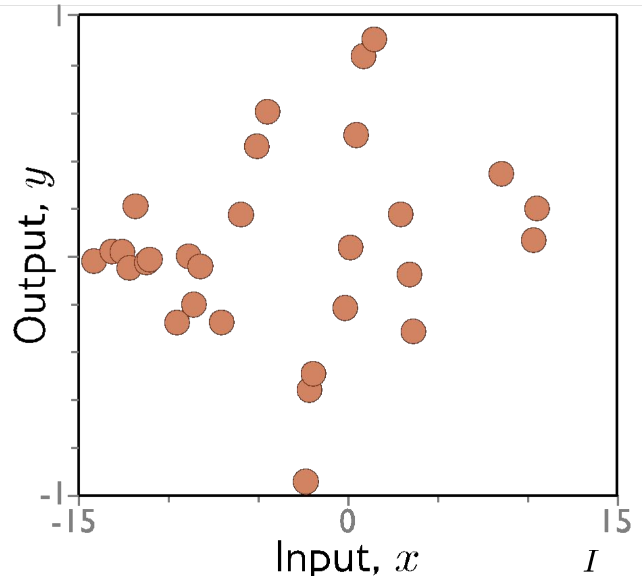


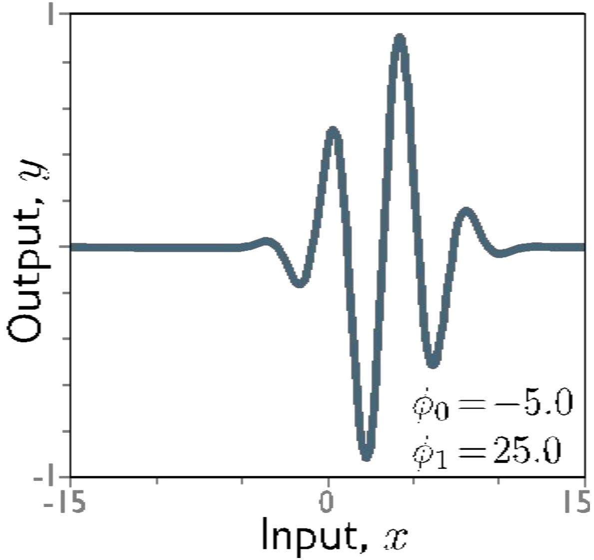
ϕ_0 shifts left and right

ϕ_1 shrinks and expands the sinusoid and envelope

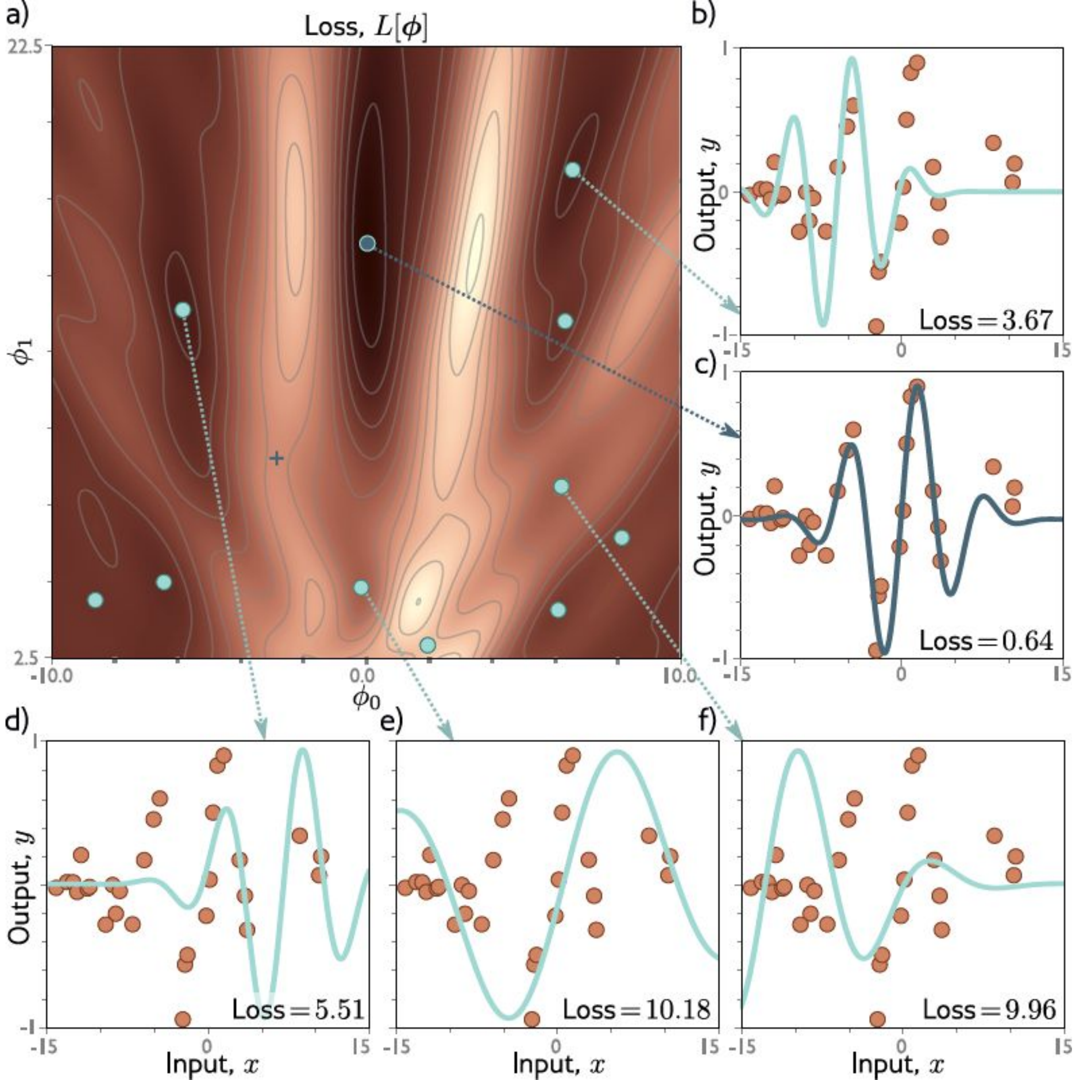
Toy Dataset and Gabor model

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

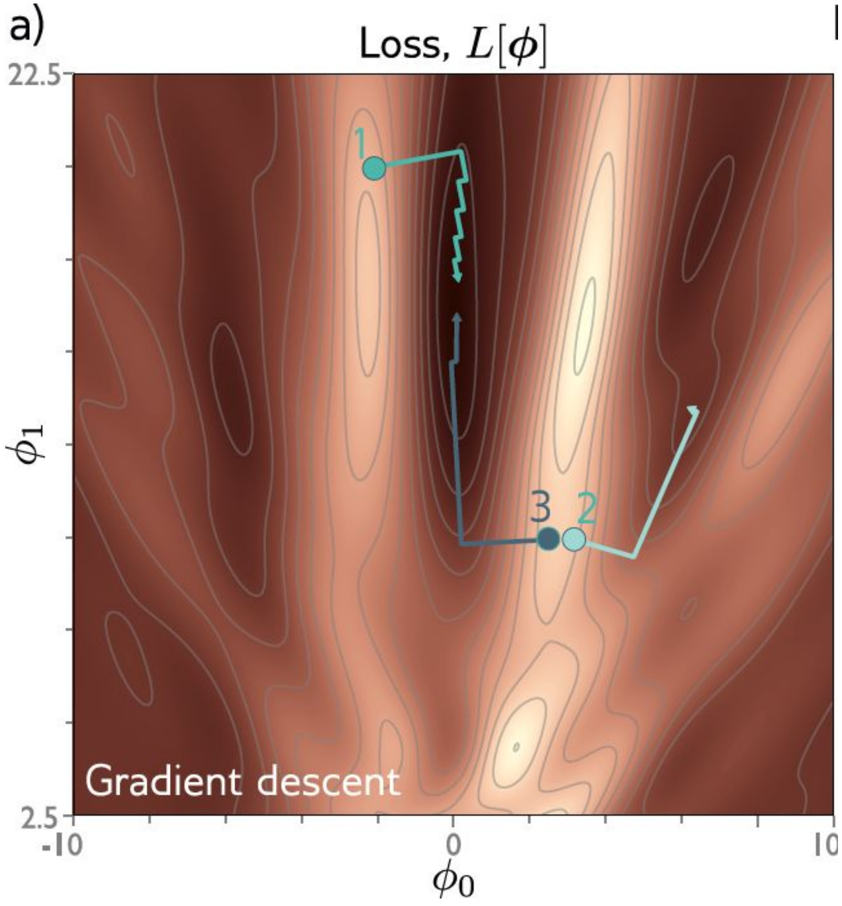




$$L[\boldsymbol{\phi}] = \sum_{i=1}^{I} (\mathrm{f}[x_i, \boldsymbol{\phi}] - y_i)^2$$



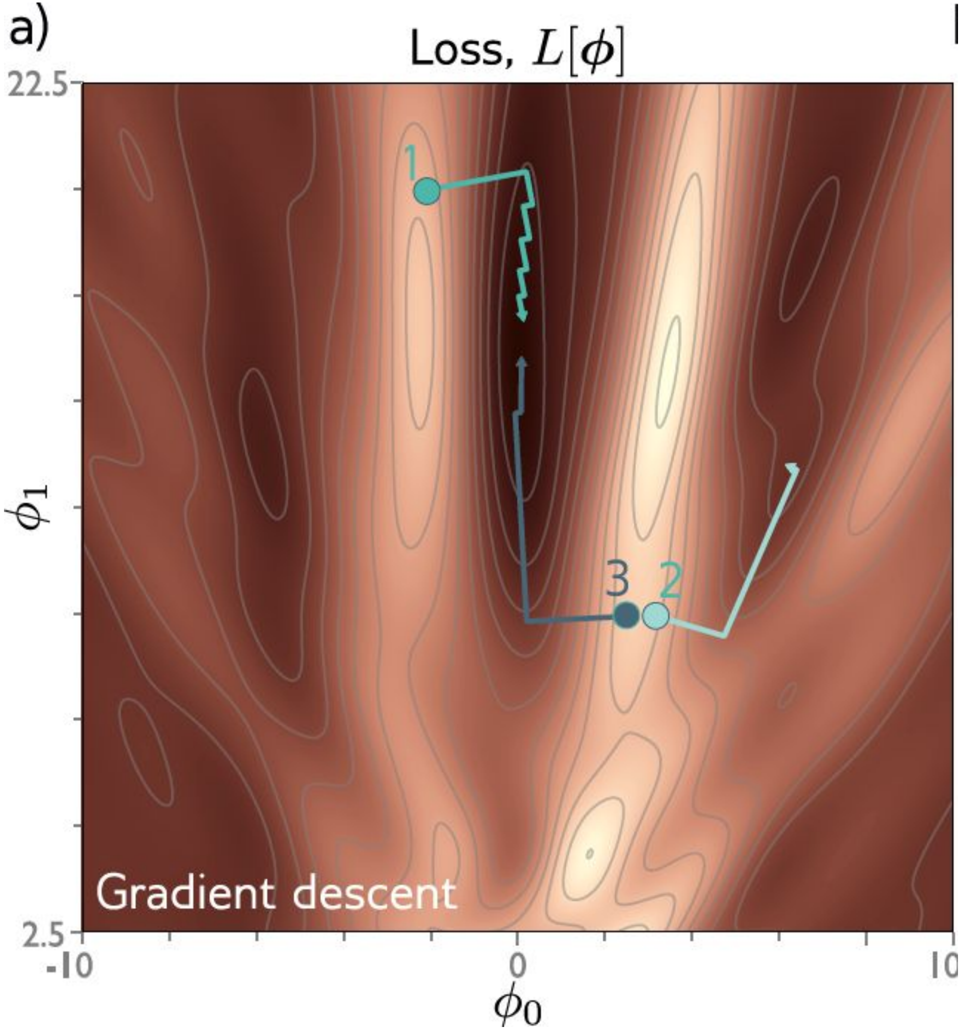
a)
Loss, $L[\phi]$
22.5
2.5
ϕ_1
-10.0
0.0
10.0
ϕ_0
b)
Loss = 3.67
c)
Loss = 0.64
d)
Loss = 5.51
e)
Loss = 10.18
f)
Loss = 9.96
Output, y
Input, x
1
0
-1
-15
0
15



- 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

- Code Preview
- 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 though the data is called an epoch

Batches and Epochs
(Ex. 30 sample dataset, batch size 5)

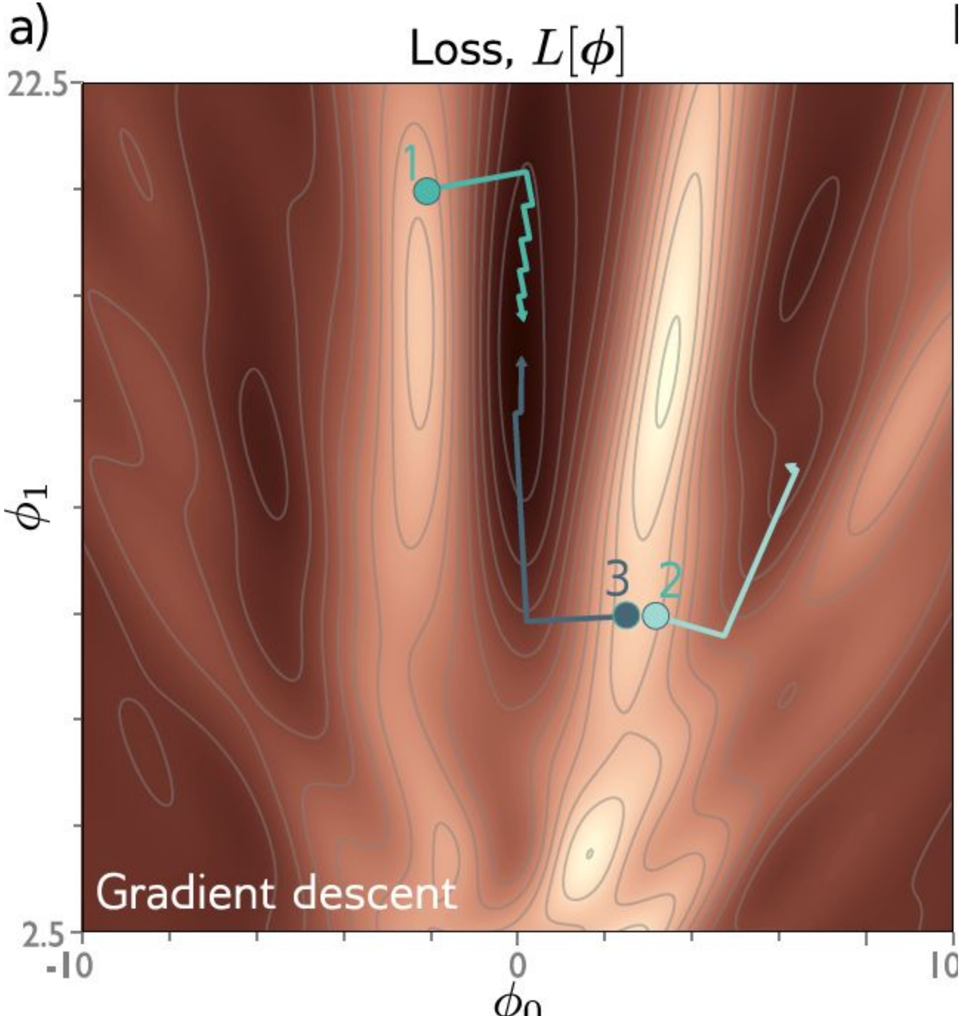
Data Indices →
```
[ 0  1  2  3 4 5  6  7  8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29]
```
Permute →
```
[27 15 23 17 8 9 28 24 12 0  4 16  5 13 11 22  1  2 25  3 21 26 18 29 20  7 10 14 19  6]
```

Batch Size 5

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]
```

…



- Stochastic gradient descent

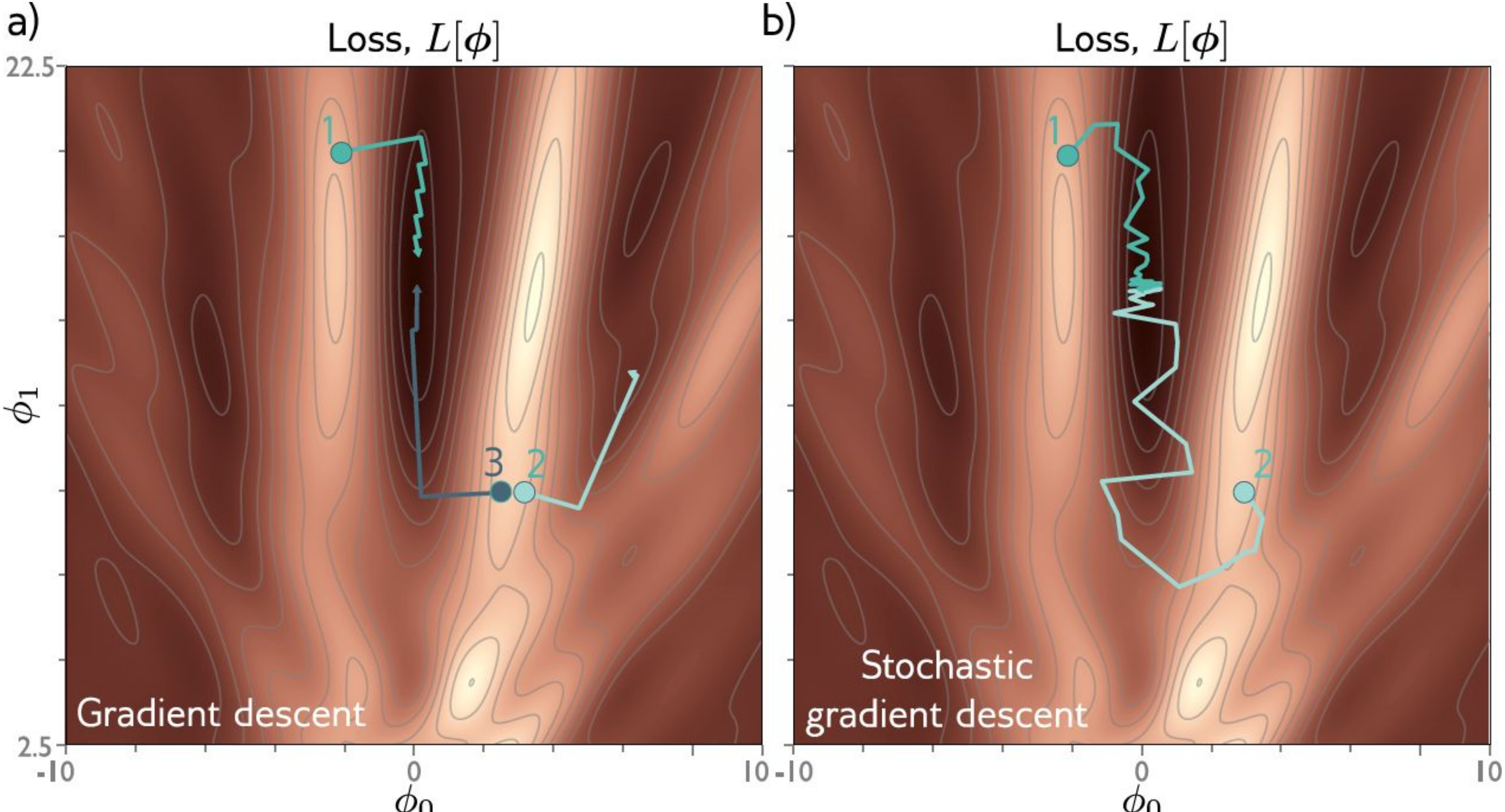
Before (full batch descent)

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

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



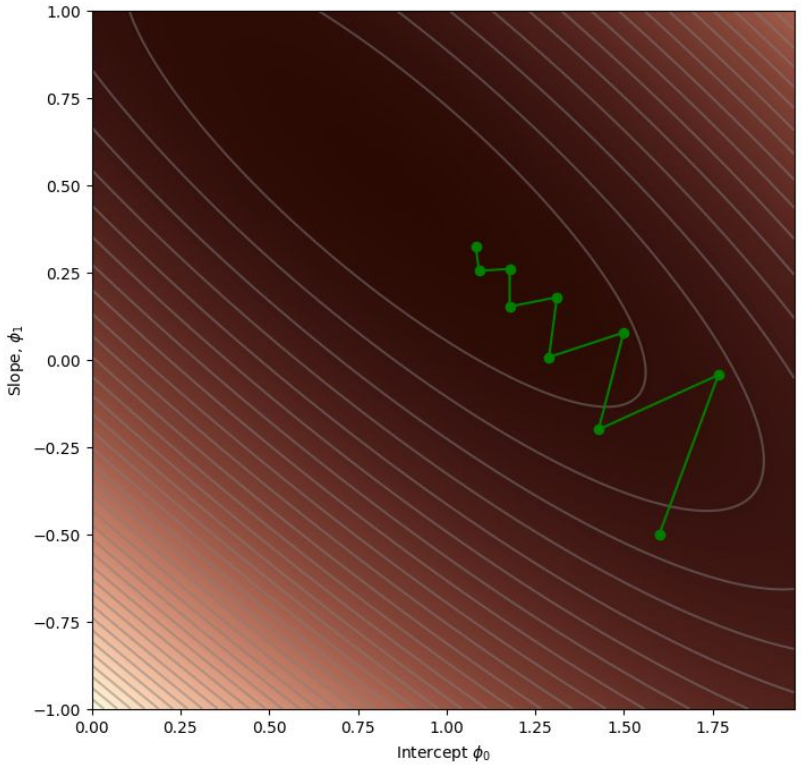
a)
Loss, $L[\phi]$
22.5
2.5
-10
0
10
ϕ_0
ϕ_1
1
2
3
Gradient descent
b)
Loss, $L[\phi]$
-10
0
10
ϕ_0
1
2
Stochastic
gradient descent

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

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

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

Momentum

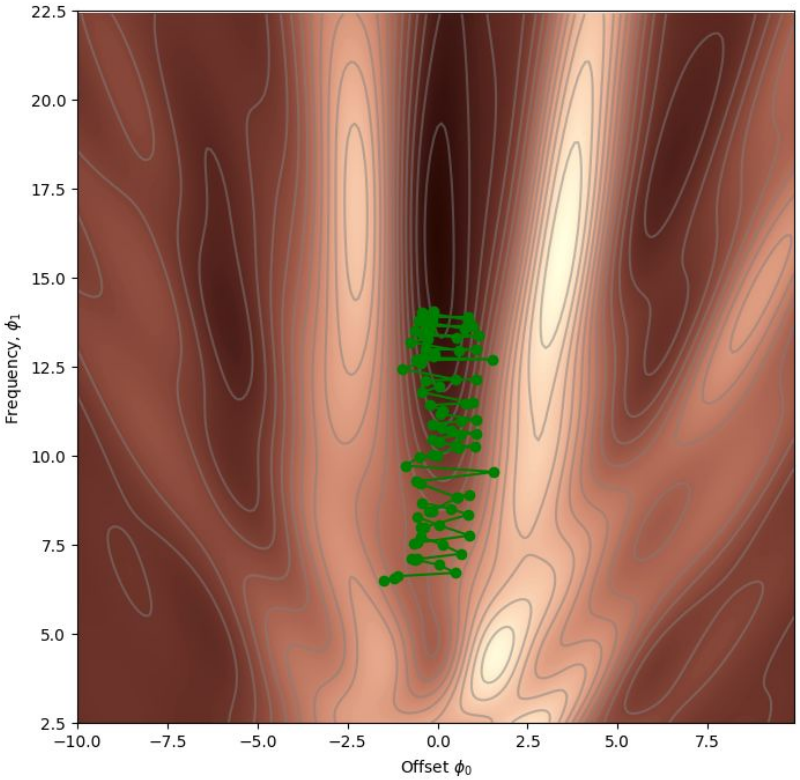
- Weighted sum of this gradient and previous gradient
- Not only influenced by gradient
- Changes more slowly over time

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

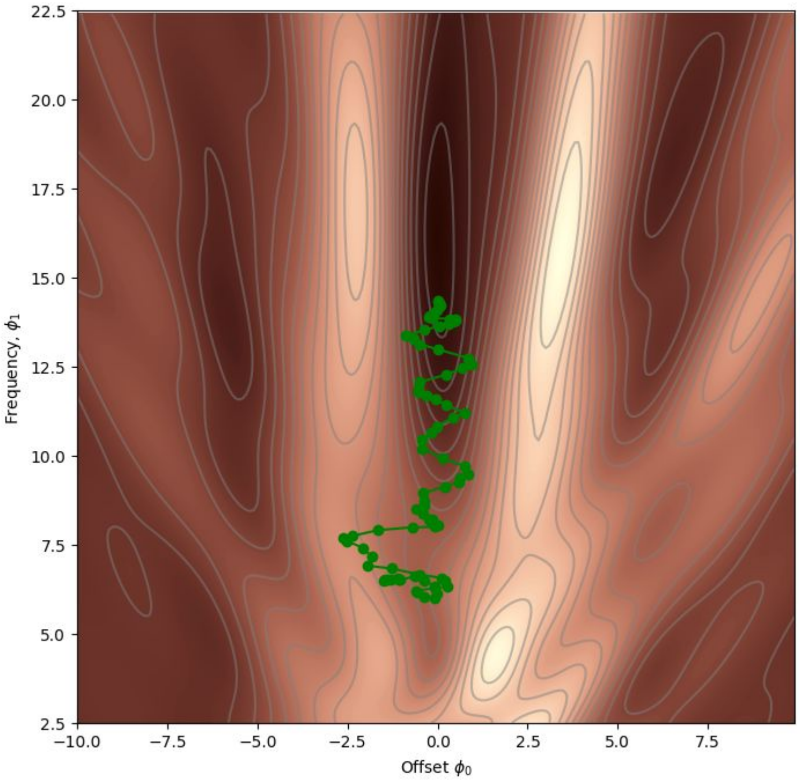
$$\boldsymbol{\phi}_{t+1} \leftarrow \boldsymbol{\phi}_t - \alpha \cdot \mathbf{m}_{t+1}$$

Still in batches.

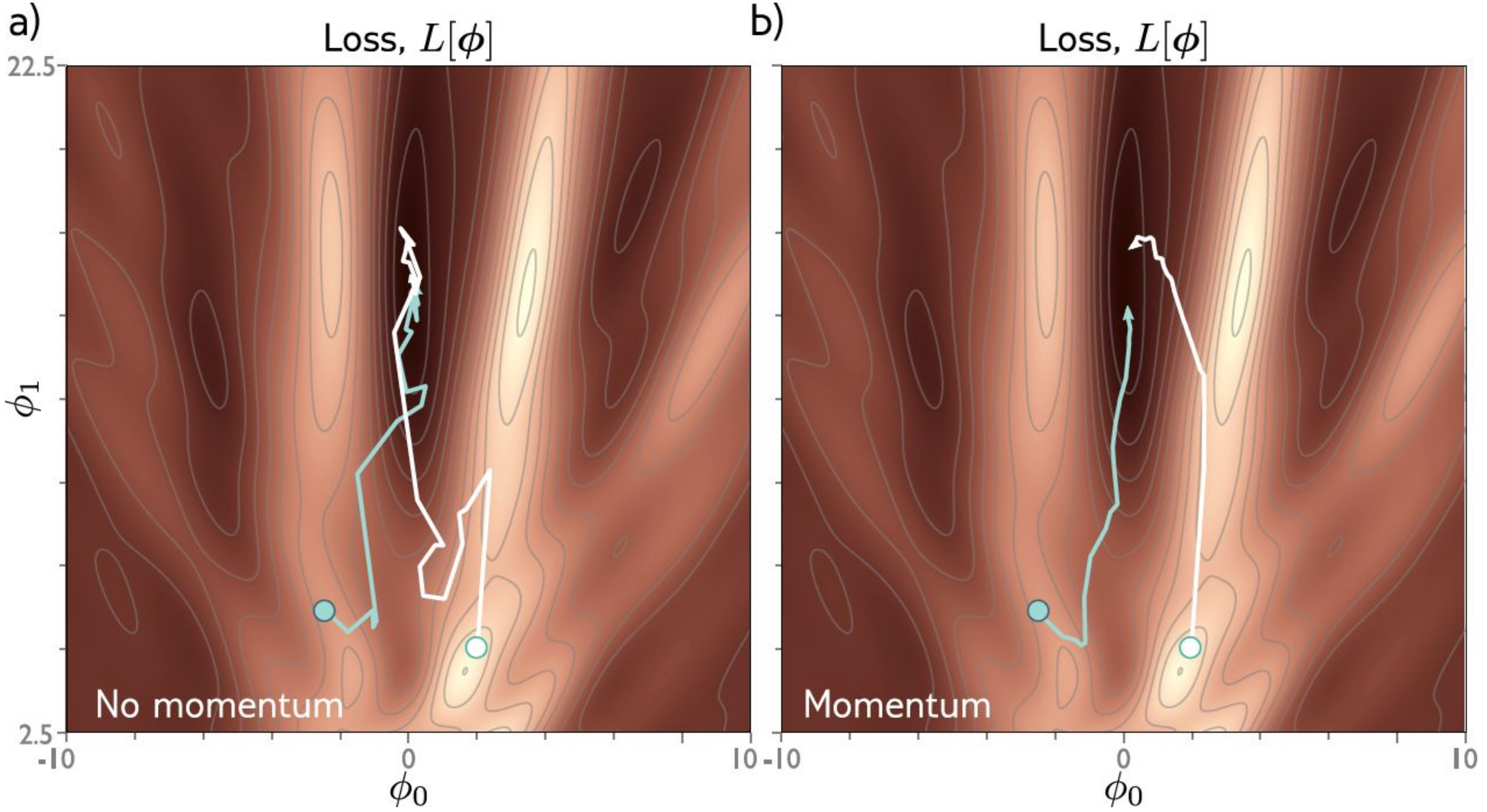
Without and With Momentum



Without Momentum, Loss = 1.31



With Momentum, Loss = 0.96



a)
Loss, $L[\phi]$
22.5
2.5
ϕ_1
-10
0
10
ϕ_0
No momentum
b)
Loss, $L[\phi]$
-10
0
10
ϕ_0
Momentum

Nesterov accelerated momentum

- Momentum smooths out gradient of current location

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

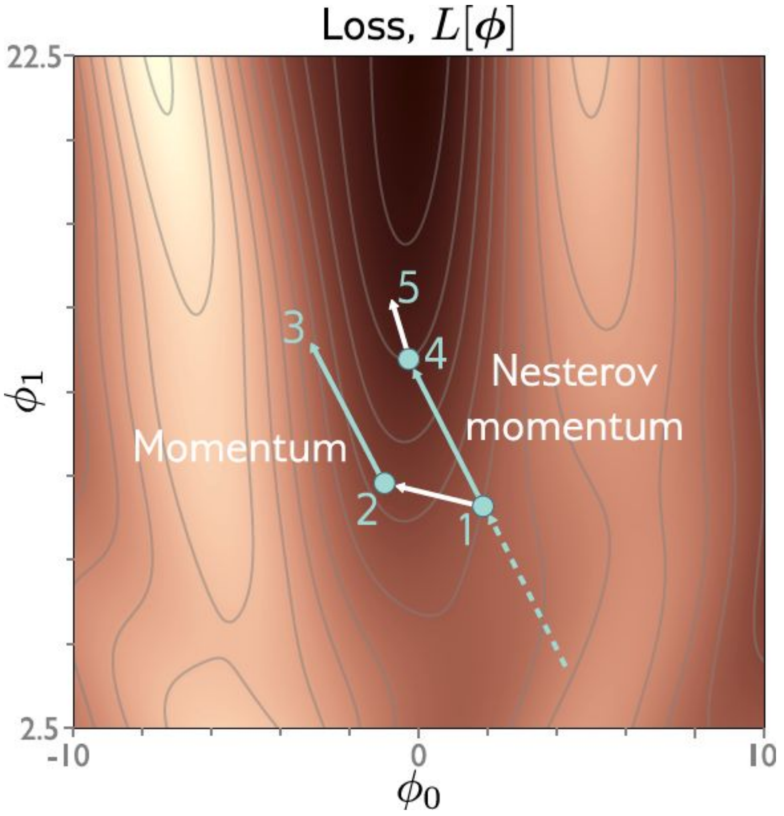
$$\boldsymbol{\phi}_{t+1} \leftarrow \boldsymbol{\phi}_t - \alpha \cdot \mathbf{m}_{t+1}$$

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

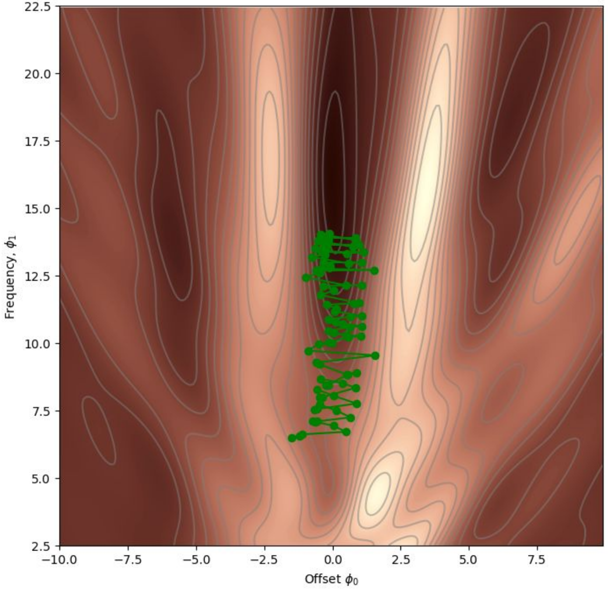
$$\mathbf{m}_{t+1} \leftarrow \beta \cdot \mathbf{m}_t + (1-\beta) \sum_{i \in \mathcal{B}_t} \frac{\partial \ell_i[\boldsymbol{\phi}_t - \alpha \cdot \mathbf{m}_t]}{\partial \boldsymbol{\phi}}$$

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

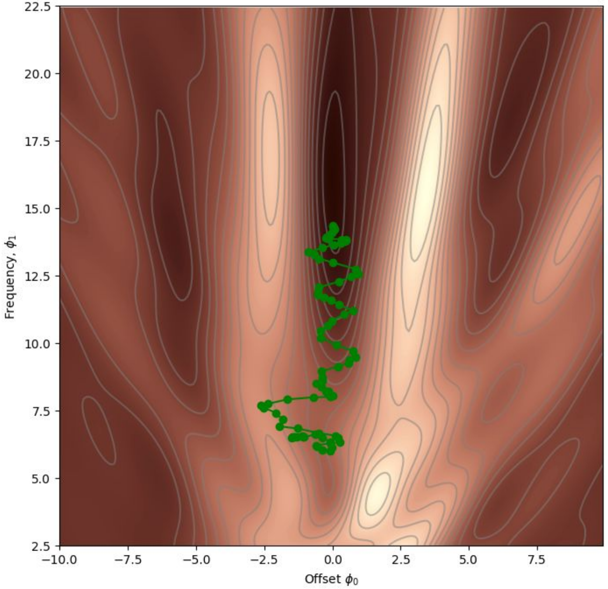
Still in batches.



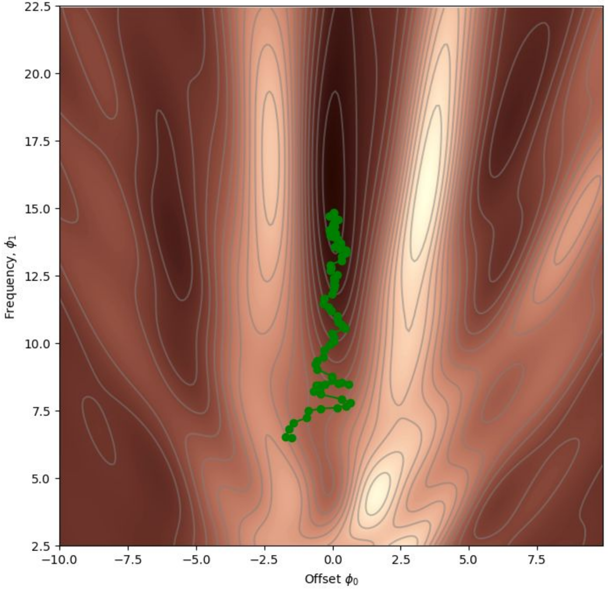
Nesterov Momentum



Without Momentum, Loss = 1.31



With Momentum, Loss = 0.96



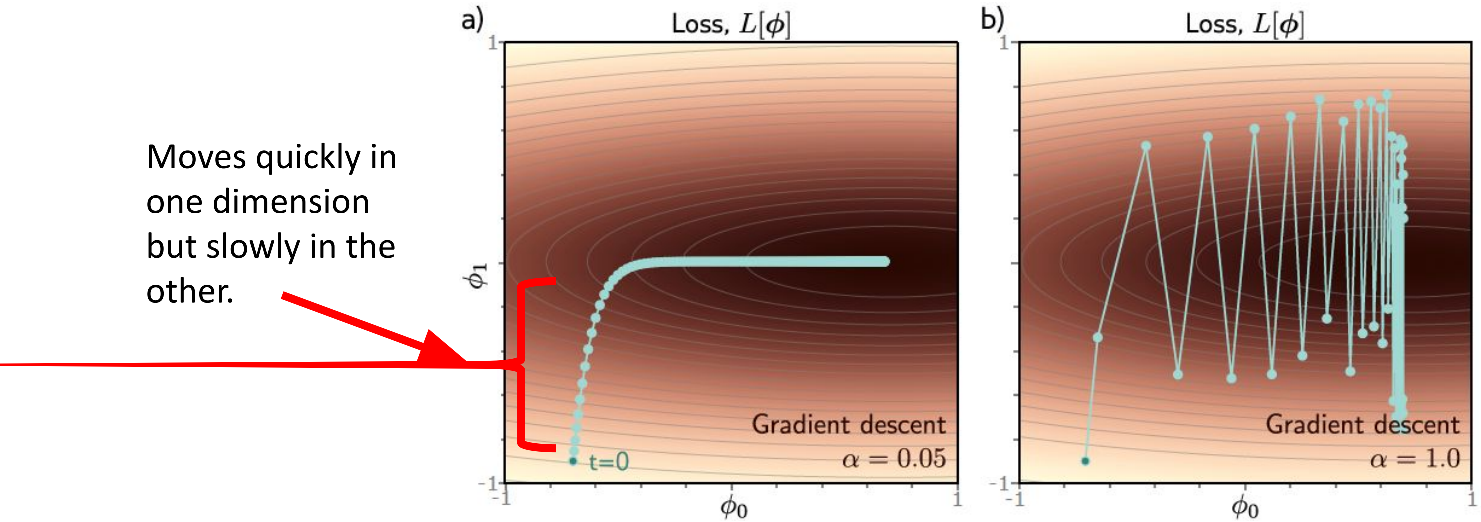
Nesterov Momentum, Loss = 0.80

Fitting models

- Code Preview
- 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 $\mathbf{v}_{t+1}$

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

- Normalize:

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

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

α is the learning rate
ϵ 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 $\mathbf{v}_{t+1}$

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

- Normalize:

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

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

α is the learning rate
ϵ 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

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

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

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

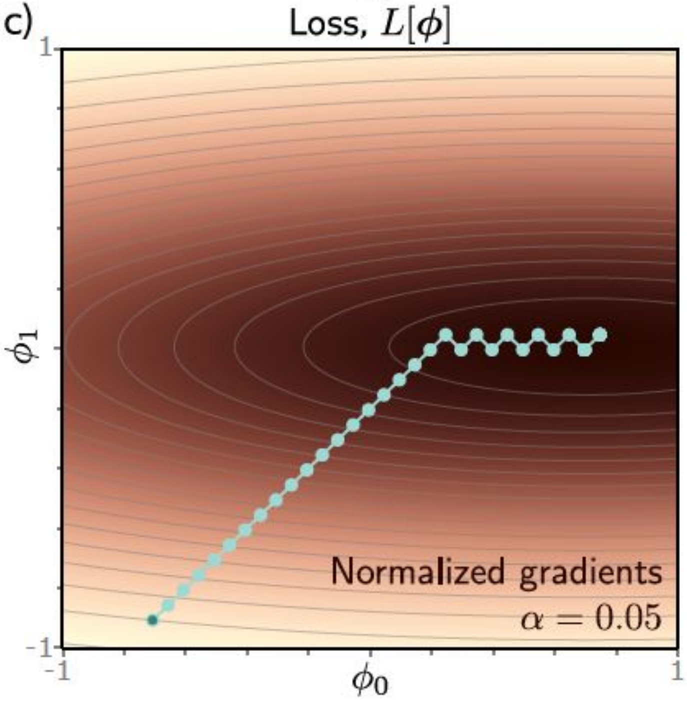
- Normalize:

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

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

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

Solution Part 1: Normalized gradients



- algorithm moves downhill a fixed distance α 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*

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

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

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

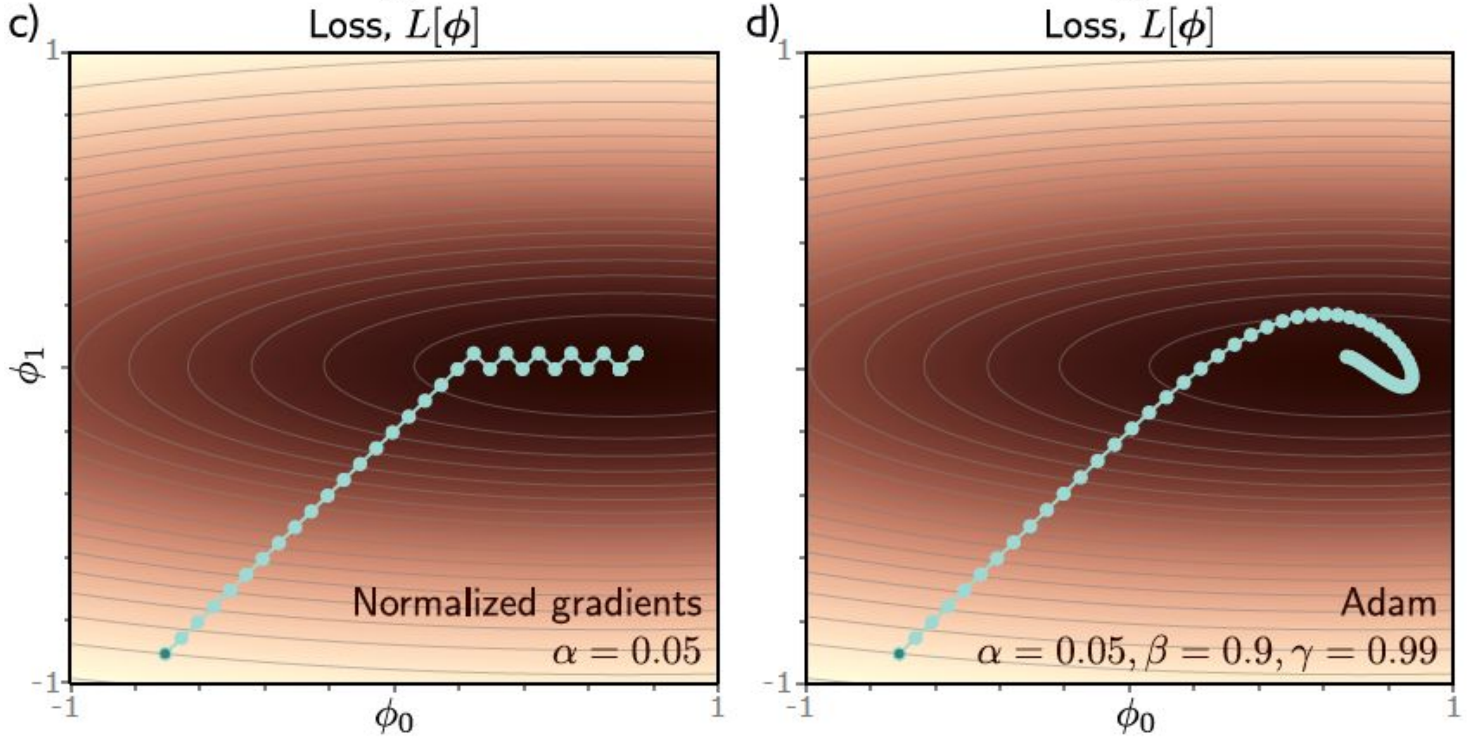
$$\tilde{\mathbf{m}}_{t+1} \leftarrow \frac{\mathbf{m}_{t+1}}{1-\beta^{t+1}} \qquad \mathbf{m}_{t=0} = 0$$

$$\tilde{\mathbf{v}}_{t+1} \leftarrow \frac{\mathbf{v}_{t+1}}{1-\gamma^{t+1}} \qquad \mathbf{v}_{t=0} = 0$$

- Update the parameters

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

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

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

Coming Up Next

- Gradients and initialization
  - Backpropagation process - efficient calculation of gradients
  - Learning rates - how aggressively do we use gradients
  - Initialization strategies - avoid bad initializations crippling learning
- Measuring Performance
  - Sounds easy - just plot losses?
  - Some subtleties to avoid overfitting
  - Some well-documented patterns where you think you are done prematurely
- Regularization
  - Tactics to reduce the generalization gap between training and test performance.
  - Often ad-hoc or heuristics to start, but slowly grounding these with theory.
- Following material will be more specific to application areas…

Feedback?