Logistic regression classifiers

CS 585, Fall 2018
Introduction to Natural Language Processing
http://people.cs.umass.edu/~miyyer/cs585/

Mohit Iyyer
College of Information and Computer Sciences
University of Massachusetts Amherst

[slides adapted from Brendan O'Connor & Jordan Boyd-Graber]

get an exercise at the front!

questions from last class….

- what is add-1 smoothing again????????????????????
- how many hours will each assignment take?
- i’m gonna miss class because of <insert reason>, how can i make up the in-class exercise that i missed?
- can you post the in-class exercise answers?
- what python version should we use for the assignments?

Logistic regression

- Log Linear Model - a.k.a. Logistic regression classifier
- Kinda like Naive Bayes, but:
  - Doesn't assume features are independent
    - Correlated features aren't overcounted
  - Discriminative training: optimize $p(y \mid text)$, not $p(y, text)$
  - Tends to work better - state of the art for doc classification, widespread hard-to-beat baseline for many tasks
  - Good off-the-shelf implementations (e.g. scikit-learn, vowpal wabbit)

Features

- Input document **d** (a string...)
- Engineer a feature function, f(d), to generate feature vector ***x***

$$f(d) \longrightarrow x$$

$$f(d) = \begin{pmatrix} \text{Count of “happy”,} \\ \text{(Count of “happy”) / (Length of doc),} \\ \text{log(1 + count of “happy”),} \\ \text{Count of “not happy”,} \\ \text{Count of words in my pre-specified} \\ \text{word list, “positive words according} \\ \text{to my favorite psychological theory”,} \\ \text{Count of “of the”,} \\ \text{Length of document,} \\ \text{...} \end{pmatrix}$$

Typically these use feature templates:
Generate many features at once

for each word w:
- ${w}_count
- ${w}_log_1_plus_count
- ${w}_with_NOT_before_it_count
- ....

- Not just word counts. Anything that might be useful!
- **Feature engineering**: when you spend a lot of time trying and testing new features. Very important!!! This is a place to put linguistics in.

step 1: featurization

1. Given an input text **X**, compute feature vector **x**

x = < *count*(nigerian), *count*(prince), *count*(nigerian prince) >

step 2: dot product w/ weights

1. Given an input text **X**, compute feature vector **x**

x = $<$ *count*(nigerian), *count*(prince), *count*(nigerian prince) $>$

2. Take dot product of **x** with weights **β** to get **z**

β = $<$ -1, -1, 4$>$

$$z = \sum_{i=0}^{|X|} \beta_i x_i$$

step 3: compute class probability

1. Given an input text **X**, compute feature vector **x**

x = < *count*(nigerian), *count*(prince), *count*(nigerian prince) >

2. Take dot product of **x** with weights **β** to get **z**

β = < -1, -1, 4>

$$z = \sum_{i=0}^{|X|} \beta_i x_i$$

3. Apply logistic function to **z**

$$P(z) = \frac{e^z}{e^z + 1} = \frac{1}{1 + e^{-z}}$$

why dot product?

$$z = \sum_{i=0}^{|X|} \beta_i x_i$$

Intuition: **weighted sum of features**

All linear models have this form!

Logistic Function

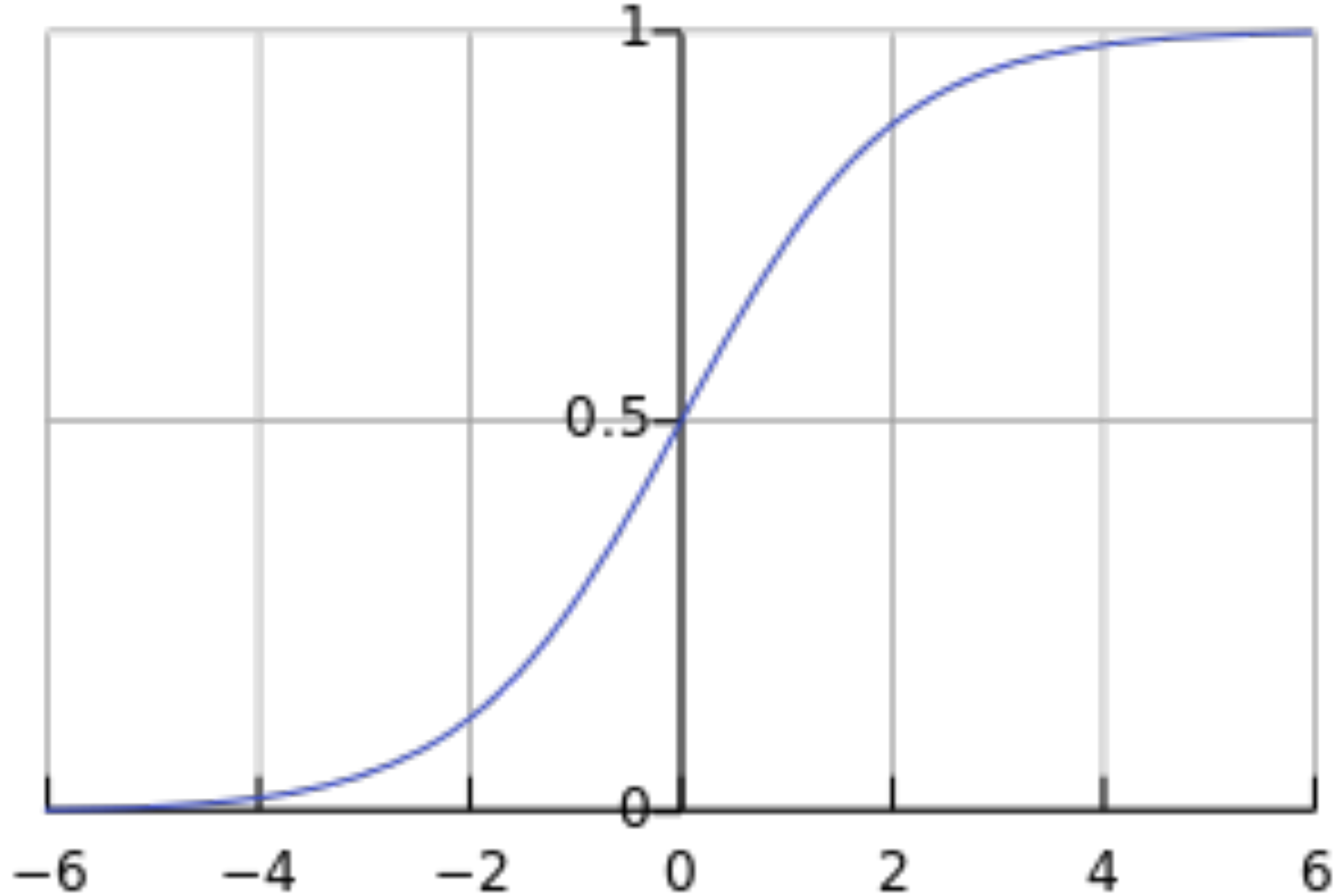
$$P(z) = \frac{e^z}{e^z + 1} = \frac{1}{1 + e^{-z}}$$

What does this function look like?

What properties does it have?

Logistic Function

$$P(z) = \frac{e^z}{e^z + 1} = \frac{1}{1 + e^{-z}}$$



Logistic Function

- logistic function $P(z) : \mathcal{R} \to [0, 1]$
- decision boundary is dot product = 0 (2 class)
- comes from linear log odds $\log \frac{P(x)}{1 - P(x)} = \sum_{i=0}^{|X|} \beta_i x_i$

How to get class probabilities?

sigmoid / logistic function: $\sigma(x) = \dfrac{1}{1 + e^{-x}}$

$$p(Y = 1 \,|\, X) = \frac{1}{1 + e^{-\sum_i \beta_i x_i}} = \frac{1}{1 + e^{-\beta x}} = \sigma(\beta x)$$

$$p(Y = 0 \,|\, X) = 1 - p(Y = 1 \,|\, X) = \frac{e^{-\beta x}}{1 + e^{-\beta x}} = 1 - \sigma(\beta x)$$

examples!

feature	coefficient	weight
bias	β_0	0.1
“viagra”	β_1	2.0
“mother”	β_2	-1.0
“work”	β_3	-0.5
“nigeria”	β_4	3.0

labels:

Y = 0 (not spam)

Y = 1 (spam)

feature	coefficient	weight
bias	β_0	0.1
"viagra"	β_1	2.0
"mother"	β_2	-1.0
"work"	β_3	-0.5
"nigeria"	β_4	3.0

labels:

Y = 0 (not spam)

Y = 1 (spam)

input 1: empty document

X = {}

$$p(Y = 1) = ???$$

$$p(Y = 0) = ???$$

feature	coefficient	weight
bias	β_0	0.1
“viagra”	β_1	2.0
“mother”	β_2	-1.0
“work”	β_3	-0.5
“nigeria”	β_4	3.0

labels:

Y = 0 (not spam)

Y = 1 (spam)

input 1: empty document

X = {}

our bias feature always fires!

$$p(Y = 1) = \frac{1}{1 + e^{-0.1}} = 0.52$$

$$p(Y = 0) = \frac{e^{-0.1}}{1 + e^{-0.1}} = 0.48$$

feature	coefficient	weight
bias	β_0	0.1
“viagra”	β_1	2.0
“mother”	β_2	-1.0
“work”	β_3	-0.5
“nigeria”	β_4	3.0

labels:

Y = 0 (not spam)

Y = 1 (spam)

input 1: empty document

X = {}

our bias feature always fires!

$$p(Y=1) = \frac{1}{1+e^{-0.1}} = 0.52$$

$$p(Y=0) = \frac{e^{-0.1}}{1+e^{-0.1}} = 0.48$$

bias encodes prior probabilities!

feature	coefficient	weight
bias	β_0	0.1
“viagra”	β_1	2.0
“mother”	β_2	-1.0
“work”	β_3	-0.5
“nigeria”	β_4	3.0

labels:

Y = 0 (not spam)

Y = 1 (spam)

input 2:

X = {mother, nigeria}

$$p(Y = 1) = ???$$

$$p(Y = 0) = ???$$

feature	coefficient	weight
bias	β_0	0.1
“viagra”	β_1	2.0
“mother”	β_2	-1.0
“work”	β_3	-0.5
“nigeria”	β_4	3.0

labels:

Y = 0 (not spam)

Y = 1 (spam)

input 2:

X = {mother, nigeria}

$$p(Y = 1) = \frac{1}{1 + e^{-(0.1 - 1.0 + 3)}} = 0.89$$

$$p(Y = 0) = 0.11$$

feature	coefficient	weight
bias	β_0	0.1
“viagra”	β_1	2.0
“mother”	β_2	-1.0
“work”	β_3	-0.5
“nigeria”	β_4	3.0

labels:

Y = 0 (not spam)

Y = 1 (spam)

input 2:

X = {mother, nigeria}

$$p(Y=1) = \frac{1}{1+e^{-(0.1-1.0+3)}} = 0.89$$

$$p(Y=0) = 0.11$$

bias + sum of other weights!

feature	**coefficient**	**weight**
bias	β_0	0.1
“viagra”	β_1	2.0
“mother”	β_2	-1.0
“work”	β_3	-0.5
“nigeria”	β_4	3.0
# tokens	β_5	4.5

labels:

Y = 0 (not spam)

Y = 1 (spam)

input 2:

X = {mother, nigeria}

what if i added a new feature for the # of tokens in the input?

feature	coefficient	weight
bias	β_0	0.1
“viagra”	β_1	2.0
“mother”	β_2	-1.0
“work”	β_3	-0.5
“nigeria”	β_4	3.0
# tokens	β_5	4.5

labels:

Y = 0 (not spam)

Y = 1 (spam)

input 2:

X = {mother, nigeria}

what if i added a new feature for the # of tokens in the input?

$$p(Y=1) = \frac{1}{1+e^{-(0.1-1.0+3+2*4.5)}}$$

NB as Log-Linear Model

- What are the **features** in Naive Bayes?

- What are the **weights** in Naive Bayes?

NB as Log-Linear Model

$$P(\text{spam}|D) \propto P(\text{spam}) \cdot \prod_{w_i \in D} P(w_i|\text{spam})$$

NB as Log-Linear Model

$$P(\text{spam}|D) \propto P(\text{spam}) \cdot \prod_{w_i \in D} P(w_i|\text{spam})$$

x_i = count of word in D

$$P(\text{spam}|D) \propto P(\text{spam}) + \prod_{w_i \in \text{Vocab}} \cdot P(w_i|\text{spam})^{x_i}$$

NB as Log-Linear Model

$$P(\text{spam}|D) \propto P(\text{spam}) \cdot \prod_{w_i \in D} P(w_i|\text{spam})$$

x$_i$ = count of word in D

$$P(\text{spam}|D) \propto P(\text{spam}) \quad \cdot \prod_{w_i \in \text{Vocab}} \cdot P(w_i|\text{spam})^{x_i}$$

$$\log[P(\text{spam}|D)] \propto \log[P(\text{spam})] + \sum_{w_i \in \text{Vocab}} x_i \cdot \log[P(w_i|\text{spam})]$$

x$_i$ are features

log probs are weights!

naive Bayes vs. logistic regression

- naive Bayes is easier to implement
- naive Bayes better on small datasets
- logistic regression better on medium-sized datasets
- on huge datasets, both perform comparably
- **biggest difference: logistic regression allows arbitrary features**

now you know everything about logistic regression except….

how do we *learn* the weights????

- in naive Bayes, we just *counted* to get conditional probabilities
- in logistic regression, we perform *stochastic gradient ascent*

Learning Weights

- given: a set of **feature vectors** and **labels**
- goal: learn the weights.

Learning Weights

We know:

$$P(z) = \frac{e^z}{e^z + 1} = \frac{1}{1 + e^{-z}}$$

So let's try to maximize probability of the entire dataset - **maximum likelihood estimation**

Learning Weights

So let's try to maximize probability of the entire dataset - **maximum likelihood estimation**

$$\beta^{MLE} = \arg\max_{\beta} \log P(y_0, \ldots, y_n | \mathbf{x_0}, \ldots, \mathbf{x_n}; \beta)$$

$$= \arg\max_{\beta} \sum_{i=0}^{|X|} \log P(y_i | \mathbf{x_i}; \beta)$$

Learning Weights

So let's try to maximize probability of the entire dataset - **maximum likelihood estimation**

$$\beta^{MLE} = \arg\max_{\beta} \log P(y_0, \ldots, y_n | \mathbf{x_0}, \ldots, \mathbf{x_n}; \beta)$$

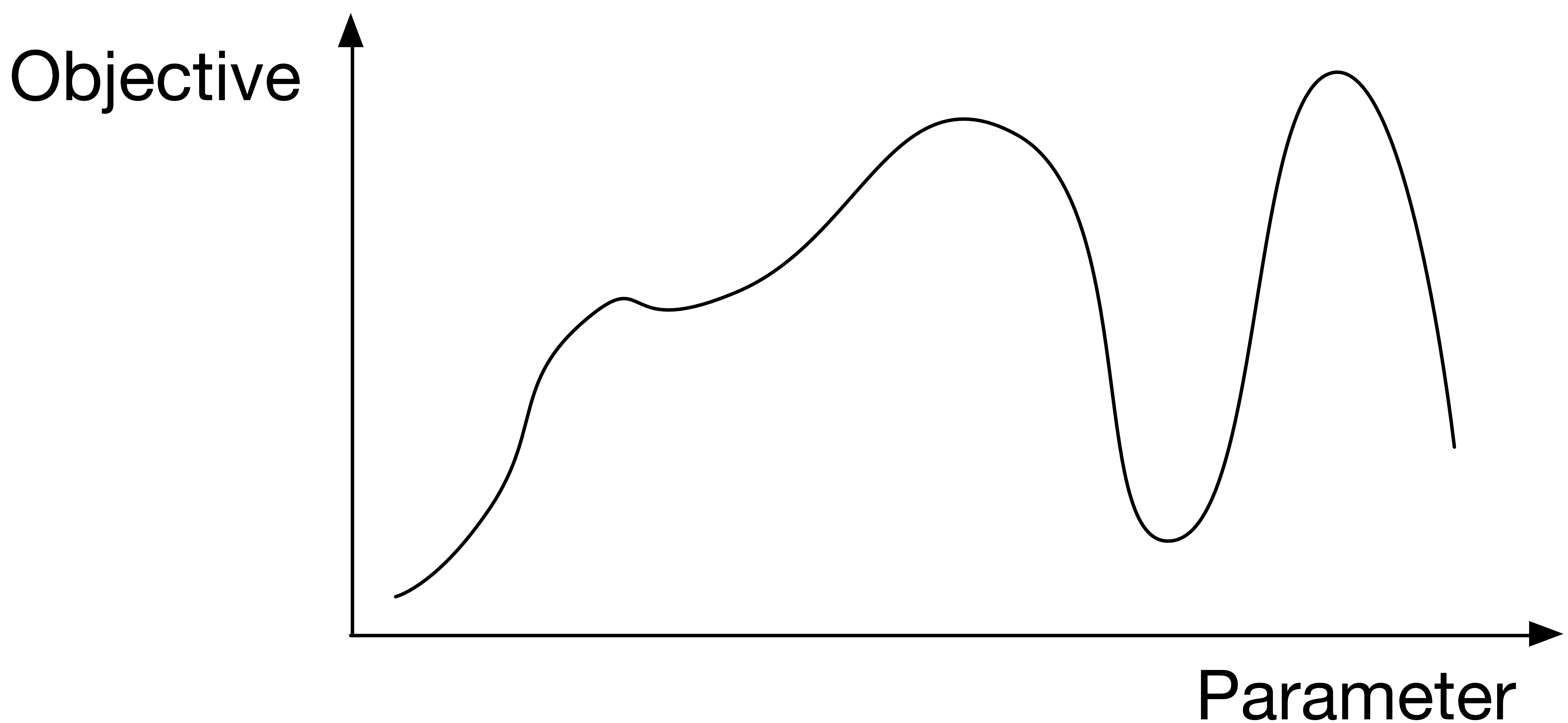
$$= \arg\max_{\beta} \sum_{i=0}^{|X|} \log P(y_i | \mathbf{x_i}; \beta)$$

equivalent to minimizing the negative log likelihood as in your reading!

gradient ascent (non-convex)

Goal

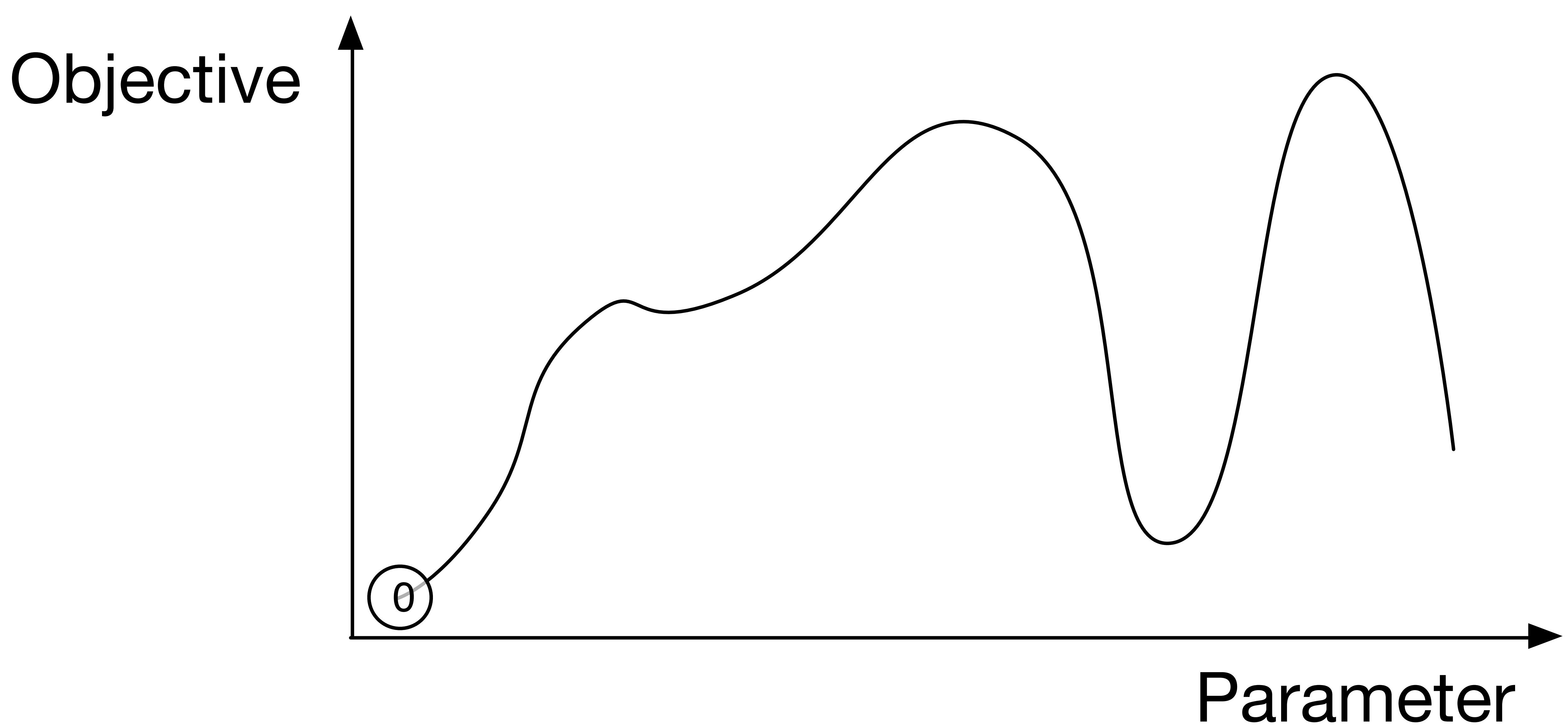
Optimize log likelihood with respect to variables β



gradient ascent (non-convex)

Goal

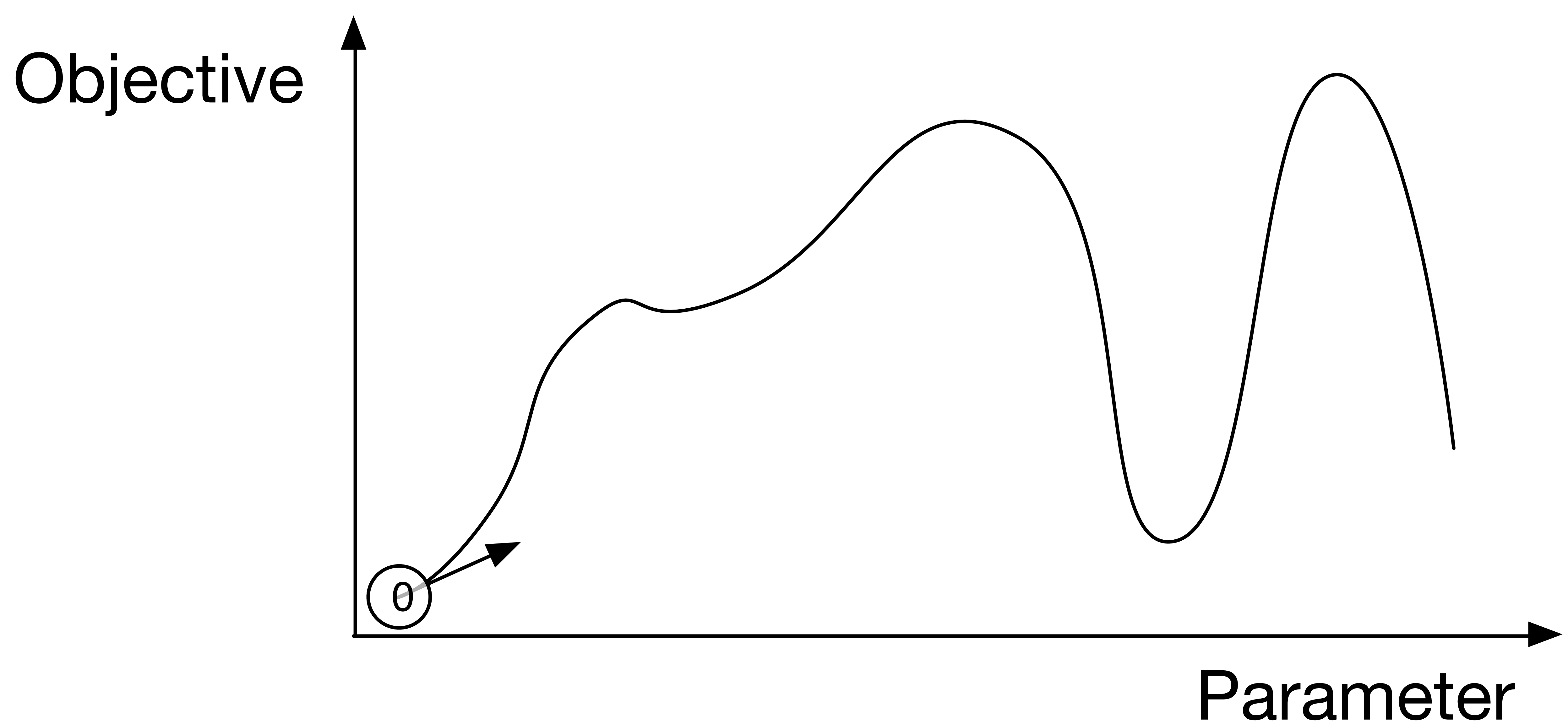
Optimize log likelihood with respect to variables β



gradient ascent (non-convex)

Goal

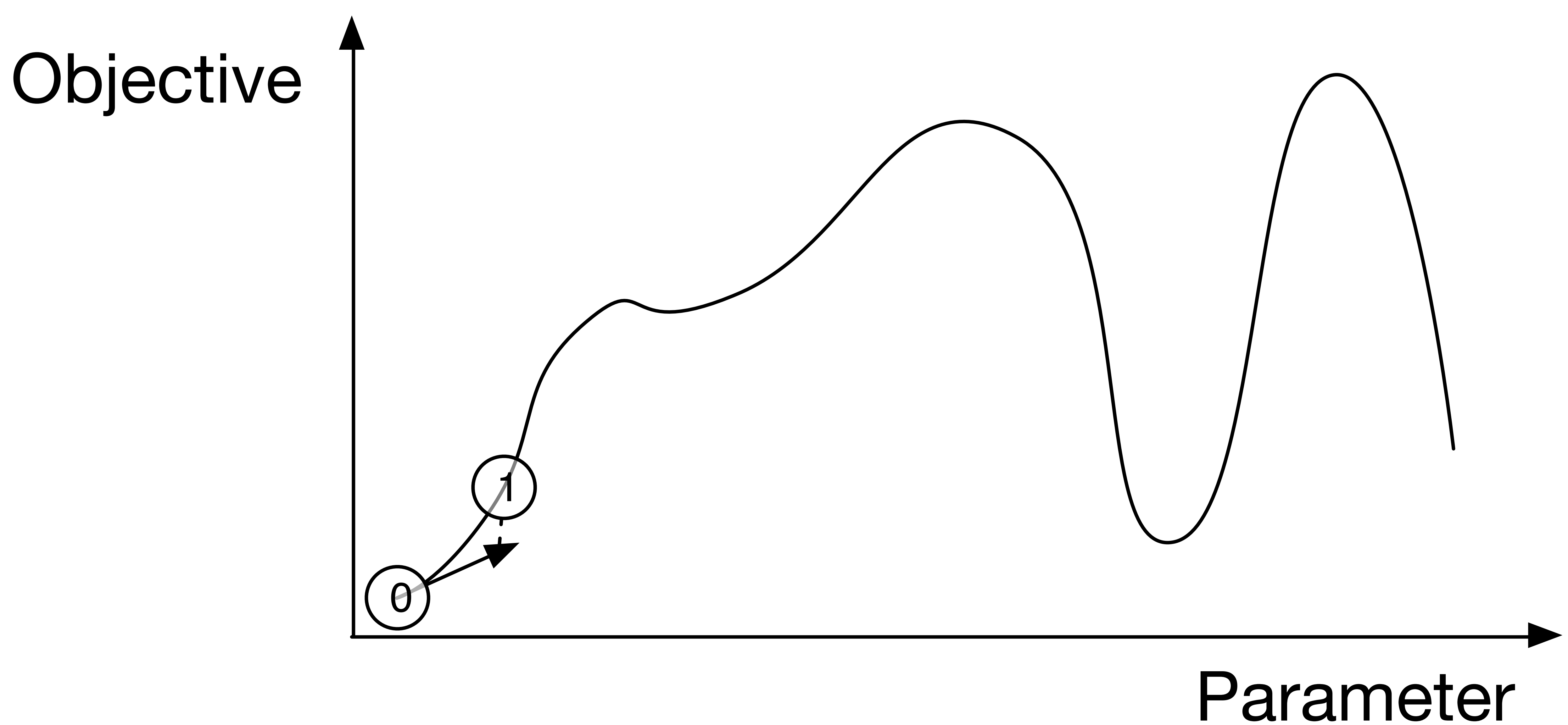
Optimize log likelihood with respect to variables β



gradient ascent (non-convex)

Goal

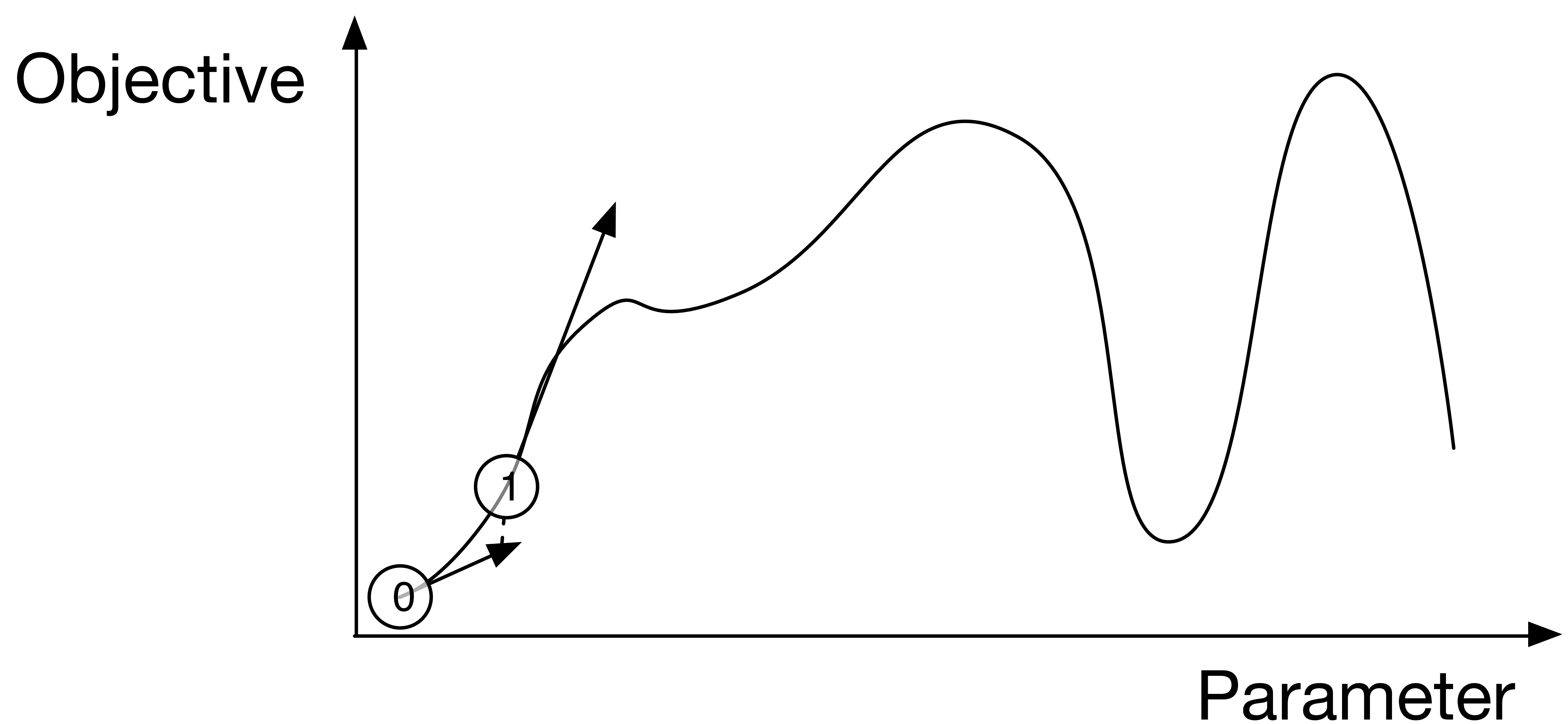
Optimize log likelihood with respect to variables β



gradient ascent (non-convex)

Goal

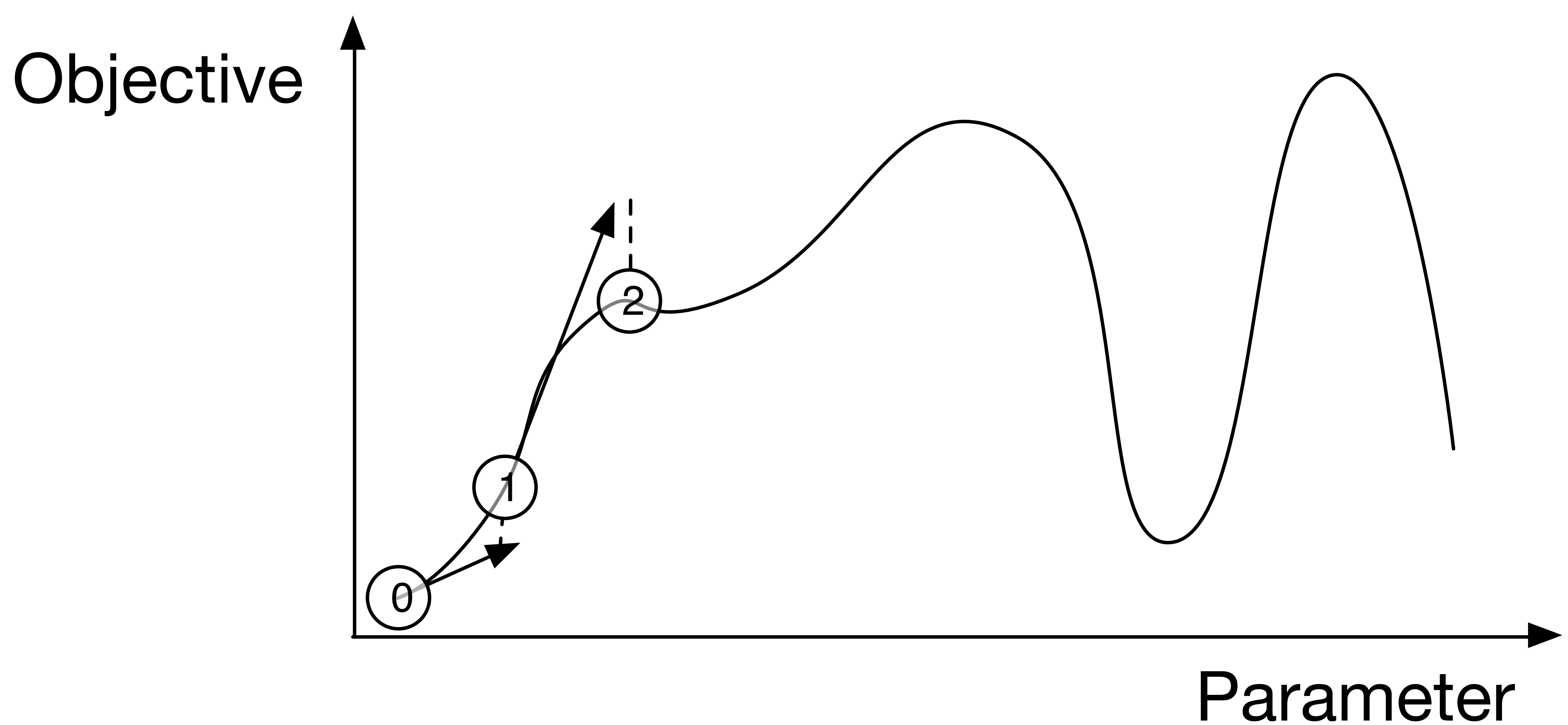
Optimize log likelihood with respect to variables β



gradient ascent (non-convex)

Goal

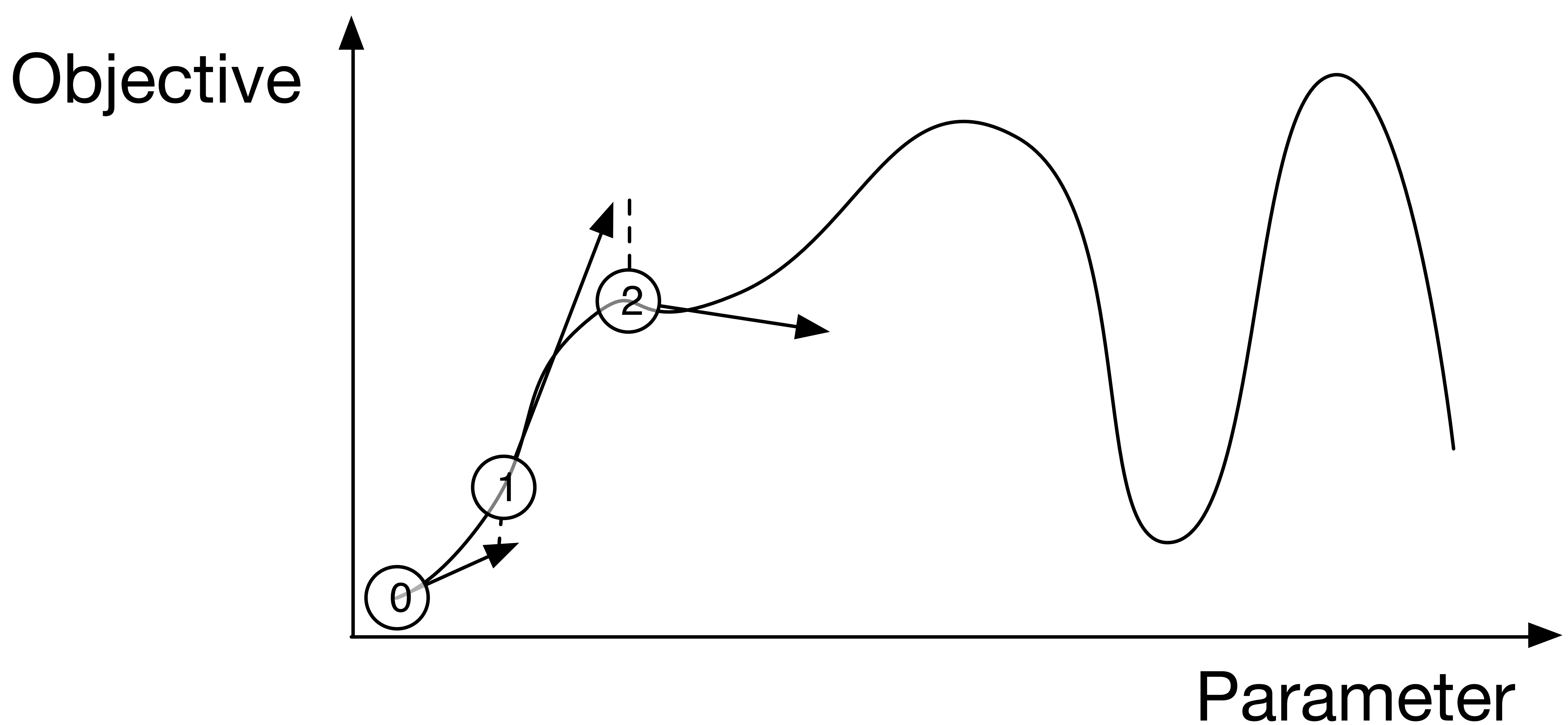
Optimize log likelihood with respect to variables β



gradient ascent (non-convex)

Goal

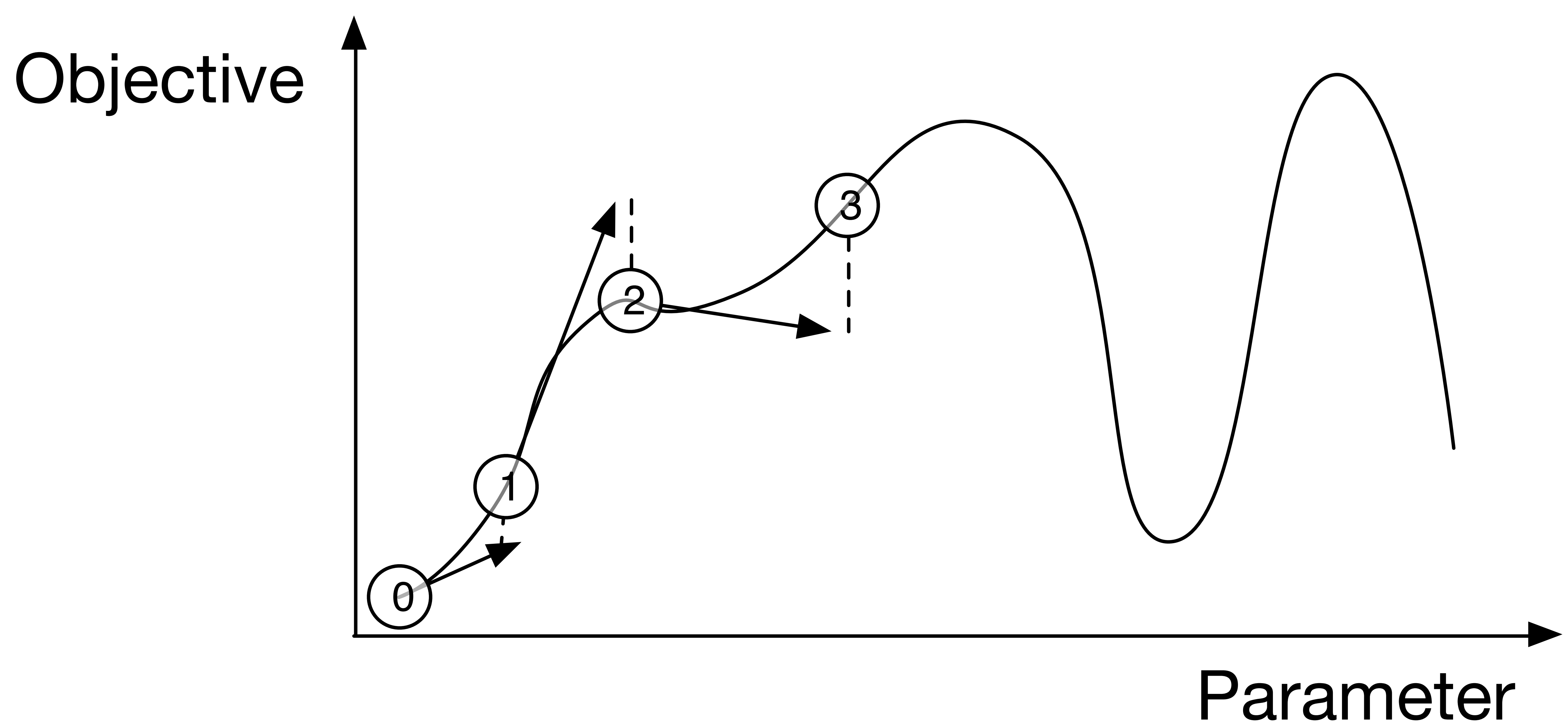
Optimize log likelihood with respect to variables β



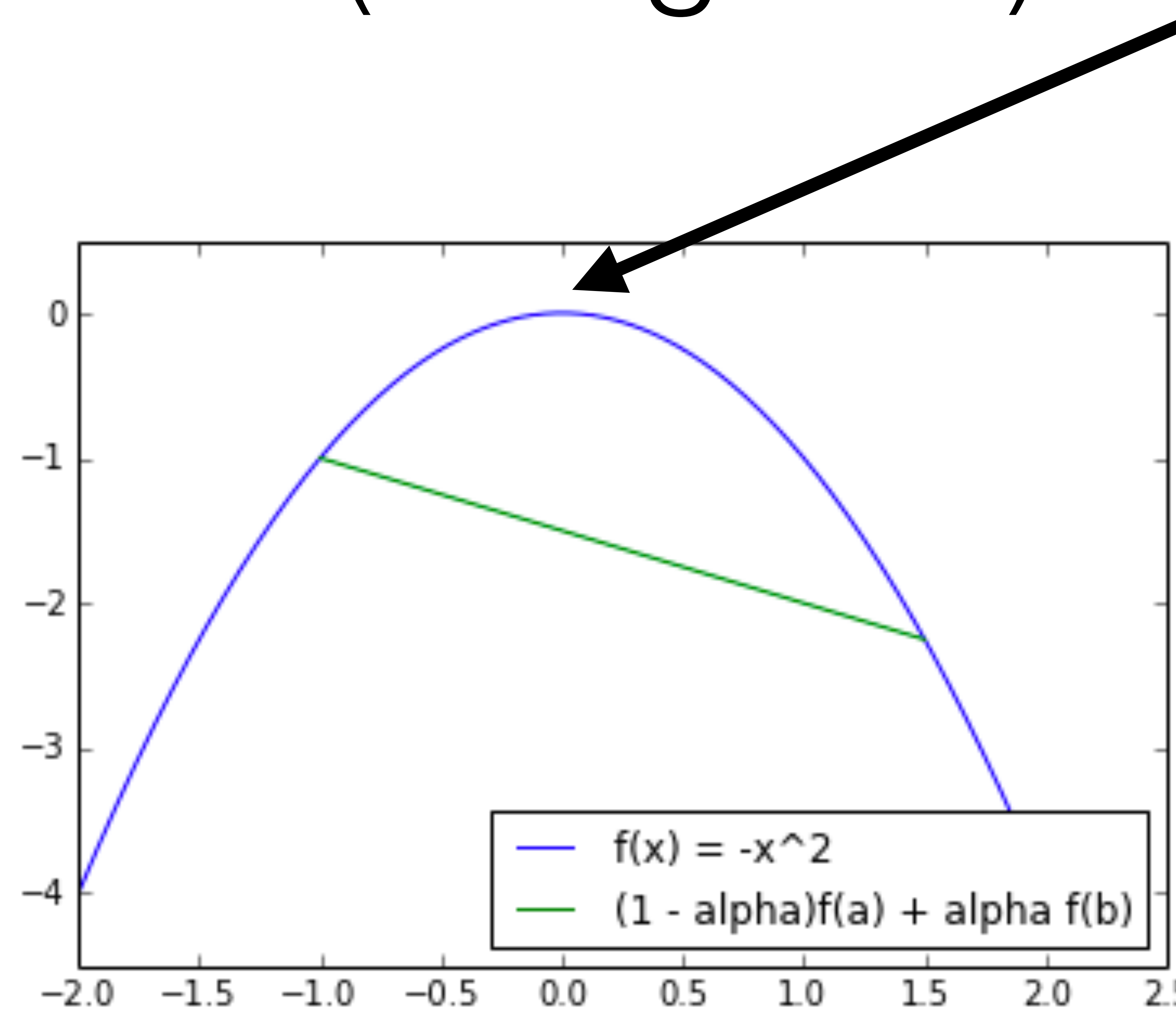
gradient ascent (non-convex)

Goal

Optimize log likelihood with respect to variables β



good news! the log-likelihood in LR is *concave*, which means that it has just one local (and global) maximum

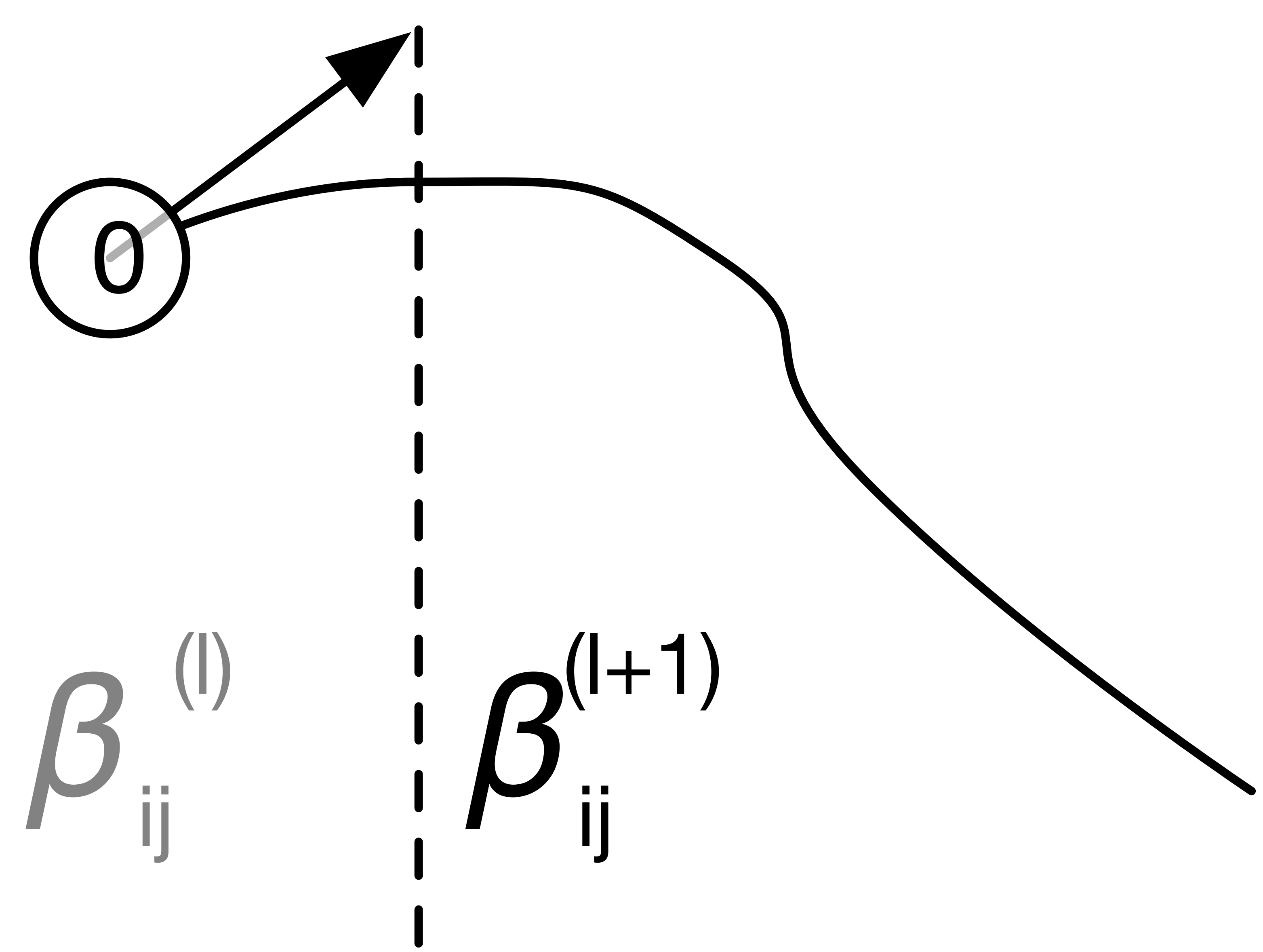


gradient ascent (non-convex)

Goal

Optimize log likelihood with respect to variables β

$$\frac{\partial \mathcal{L}}{\partial \beta} = \text{gradient}$$



Gradient for Logistic Regression

To ease notation, let's define

$$\pi_i = \sigma(\beta \cdot x_i)$$

Our objective function is

$$\mathcal{L} = \sum_i \log p(y_i \mid x_i) = \sum_i \mathcal{L}_i = \sum_i \begin{cases} \log \pi_i & \text{if } y_i = 1 \\ \log(1 - \pi_i) & \text{if } y_i = 0 \end{cases}$$

log likelihood!

Taking the Derivative

$$\boxed{\beta_j = j^{\text{th}} \text{ dimension of } \beta}$$

$$\boxed{\frac{\partial}{\partial x} log(x) = \frac{1}{x}}$$

Apply chain rule:

$$\frac{\partial \mathscr{L}}{\partial \beta_j} = \sum_i \frac{\partial \mathscr{L}_i(\vec{\beta})}{\partial \beta_j} = \sum_i \begin{cases} \frac{1}{\pi_i} \frac{\partial \pi_i}{\partial \beta_j} & \text{if } y_i = 1 \\ \frac{1}{1-\pi_i} \left(-\frac{\partial \pi_i}{\partial \beta_j} \right) & \text{if } y_i = 0 \end{cases}$$

Taking the Derivative

$$\boxed{\beta_j = j^{\text{th}} \text{ dimension of } \beta}$$

$$\boxed{\frac{\partial}{\partial x} log(x) = \frac{1}{x}}$$

Apply chain rule:

$$\frac{\partial \mathscr{L}}{\partial \beta_j} = \sum_i \frac{\partial \mathscr{L}_i(\vec{\beta})}{\partial \beta_j} = \sum_i \begin{cases} \frac{1}{\pi_i} \frac{\partial \pi_i}{\partial \beta_j} & \text{if } y_i = 1 \\ \frac{1}{1-\pi_i}\left(-\frac{\partial \pi_i}{\partial \beta_j}\right) & \text{if } y_i = 0 \end{cases}$$

If we plug in the derivative,

$$\frac{\partial \pi_i}{\partial \beta_j} = \pi_i(1 - \pi_i)x_{i_j}$$

$$\boxed{\frac{\partial}{\partial x}\sigma(x) = \sigma(x)(1 - \sigma(x))}$$

Taking the Derivative

$\boxed{\beta_j = j^{\text{th}} \text{ dimension of } \beta}$

$$\boxed{\frac{\partial}{\partial x} log(x) = \frac{1}{x}}$$

Apply chain rule:

$$\frac{\partial \mathcal{L}}{\partial \beta_j} = \sum_i \frac{\partial \mathcal{L}_i(\vec{\beta})}{\partial \beta_j} = \sum_i \begin{cases} \frac{1}{\pi_i} \frac{\partial \pi_i}{\partial \beta_j} & \text{if } y_i = 1 \\ \frac{1}{1-\pi_i} \left(-\frac{\partial \pi_i}{\partial \beta_j} \right) & \text{if } y_i = 0 \end{cases}$$

If we plug in the derivative,

$$\frac{\partial \pi_i}{\partial \beta_j} = \pi_i (1 - \pi_i) x_{i_j}$$

$$\boxed{\frac{\partial}{\partial x} \sigma(x) = \sigma(x)(1 - \sigma(x))}$$

we can merge these two cases

$$\frac{\partial \mathcal{L}_i}{\partial \beta_j} = (y_i - \pi_i) x_{i_j}$$

Taking the Derivative

$\boxed{\beta_j = j^{\text{th}} \text{ dimension of } \beta}$

$$\boxed{\frac{\partial}{\partial x} log(x) = \frac{1}{x}}$$

Apply chain rule:

$$\frac{\partial \mathcal{L}}{\partial \beta_j} = \sum_i \frac{\partial \mathcal{L}_i(\vec{\beta})}{\partial \beta_j} = \sum_i \begin{cases} \frac{1}{\pi_i} \frac{\partial \pi_i}{\partial \beta_j} & \text{if } y_i = 1 \\ \frac{1}{1-\pi_i} \left(-\frac{\partial \pi_i}{\partial \beta_j}\right) & \text{if } y_i = 0 \end{cases}$$

If we plug in the derivative,

$$\frac{\partial \pi_i}{\partial \beta_j} = \pi_i(1 - \pi_i) x_{i_j}$$

$$\boxed{\frac{\partial}{\partial x} \sigma(x) = \sigma(x)(1 - \sigma(x))}$$

we can merge these two cases

y_i = ground-truth label

$$\frac{\partial \mathcal{L}_i}{\partial \beta_j} = (y_i - \pi_i) x_{i_j}$$

π_i = predicted probability

Gradient for Logistic Regression

Gradient

$$\nabla_\beta \mathcal{L}(\vec{\beta}) = \left[\frac{\partial \mathcal{L}(\vec{\beta})}{\partial \beta_0}, \ldots, \frac{\partial \mathcal{L}(\vec{\beta})}{\partial \beta_n} \right]$$

gradient = partial derivative of log likelihood WRT each weight

Update

$$\Delta\beta \equiv \eta \nabla_\beta \mathcal{L}(\vec{\beta})$$

$$\beta_i' \leftarrow \beta_i + \eta \frac{\partial \mathcal{L}(\vec{\beta})}{\partial \beta_i}$$

η is the *learning rate*

LogReg Exercise

features: (count "nigerian", count "prince", count "nigerian prince")

$\beta^{(0)}$ = (1.0, -3.0, 2.0) → **63% accuracy**

LogReg Exercise

features: (count "nigerian", count "prince", count "nigerian prince")

$\beta^{(0)}$ = (1.0, -3.0, 2.0) → **63% accuracy**

$\beta^{(1)}$ = (0.5, -1.0, 3.0) → **75% accuracy**

LogReg Exercise

features: (count “nigerian”, count “prince”, count “nigerian prince”)

$\beta^{(0)}$ = (1.0, -3.0, 2.0) → **63% accuracy**

$\beta^{(1)}$ = (0.5, -1.0, 3.0) → **75% accuracy**

$\beta^{(2)}$ = (-1.0, -1.0, 4.0) → **81% accuracy**

Regularized Conditional Log Likelihood

Unregularized

$$\beta^* = \arg\max_{\beta} \ln\left[p(y^{(j)} \mid x^{(j)}, \beta)\right]$$

Regularized

$$\beta^* = \arg\max_{\beta} \ln\left[p(y^{(j)} \mid x^{(j)}, \beta)\right] - \mu \sum_i \beta_i^2$$

μ is “regularization” parameter that trades off between likelihood and having small parameters

exercise!