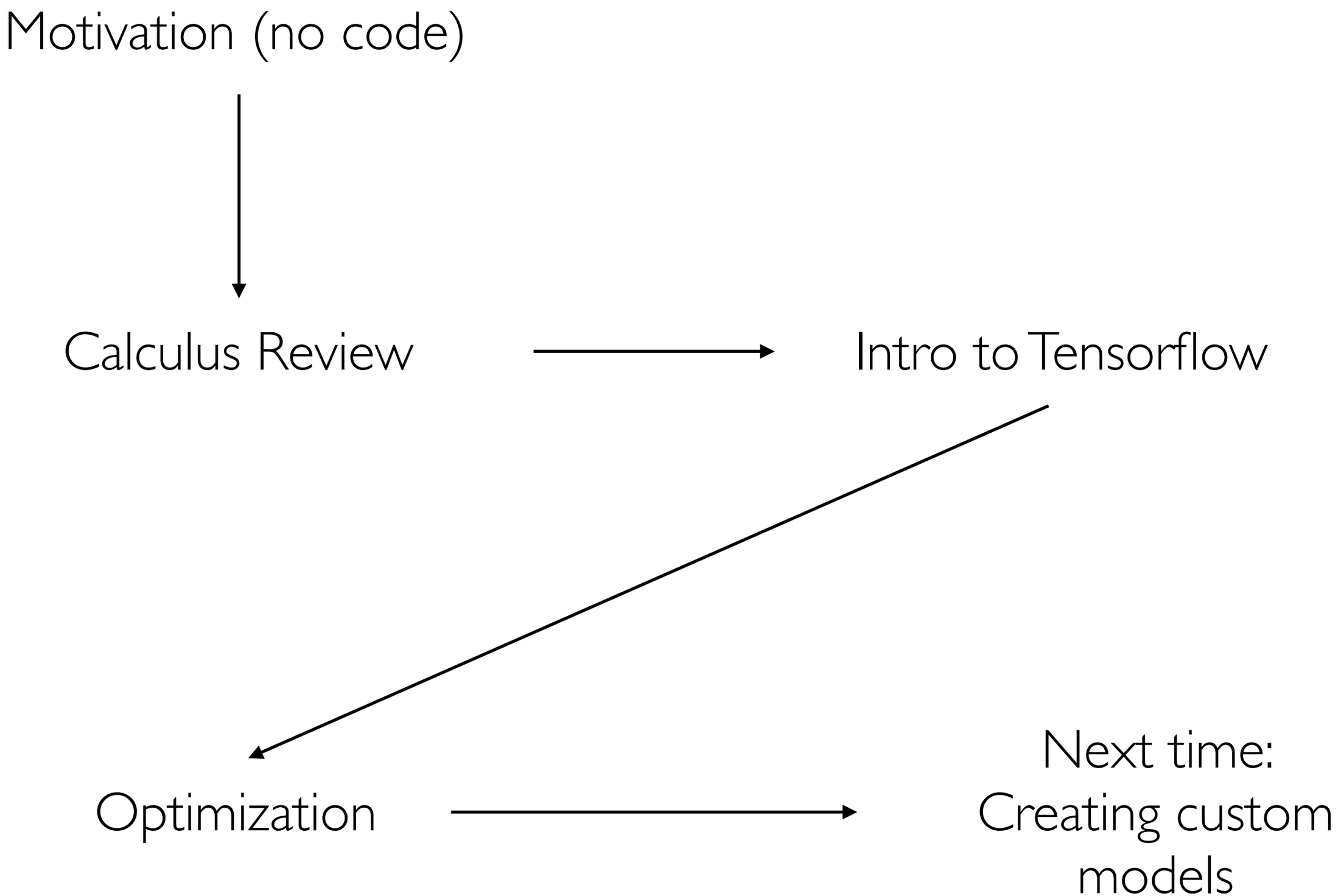
Psych 253
Advanced Statistical Modeling

Optimization

Daniel Yamins
Wu Tsai Neurosciences Institute
Departments of Psychology and Computer Science
Stanford Artificial Intelligence Laboratory
Stanford University

Russ Poldrack
Department of Psychology
Stanford University

Plan for next two class sessions



Plan for next two class sessions

Motivation (no code) *these slides*

Calculus Review
calc review slides & code notebook

tensorflow intro slides & code notebook
Intro to Tensorflow

Optimization
these slides, optimization code notebook

Next time: Creating custom models

Motivation

Why do I want to teach you about Optimization?

Motivation

Why do I want to teach you about Optimization?

Neural system identification for large populations separating "what" and "where"

David A. Klindt, Alexander S. Ecker, Thomas Euler, Matthias Bethge

(Submitted on 7 Nov 2017 (v1), last revised 29 Jan 2018 (this version, v2))

Neuroscientists classify neurons into different types that perform similar computations at different locations in the visual field. Traditional methods for neural system identification do not capitalize on this separation of 'what' and 'where'. Learning deep convolutional feature spaces that are shared among many neurons provides an exciting path forward, but the architectural design needs to account for data limitations: While new experimental techniques enable recordings from thousands of neurons, experimental time is limited so that one can sample only a small fraction of each neuron's response space. Here, we show that a major bottleneck for fitting convolutional neural networks (CNNs) to neural data is the estimation of the individual receptive field locations, a problem that has been scratched only at the surface thus far. We propose a CNN architecture with a sparse readout layer factorizing the spatial (where) and feature (what) dimensions. Our network scales well to thousands of neurons and short recordings and can be trained end-to-end. We evaluate this architecture on ground-truth data to explore the challenges and limitations of CNN-based system identification. Moreover, we show that our network model outperforms current state-of-the art system identification models of mouse primary visual cortex.

Motivation

Why do I want to teach you about Optimization?

Neural system identification for large populations separating "what" and "where"

David A. Klindt, Alexander S. Ecker. Thomas Euler. Matthias Bethge

(Submitted on 7 Nov 2

Neuroscientists cl
methods for neur
spaces that are sh
limitations: While
sample only a sm
networks (CNNs) t
surface thus far. V
Our network scale
ground-truth data
model outperform

Traditional
l feature
for data
that one can
neural
only at the
limensions.
cture on
network

An Empirical Model for Strategic Network Formation

Nicholas A. Christakis, James H. Fowler, Guido W. Imbens, Karthik Kalyanaraman

NBER Working Paper No. 16039
Issued in May 2010
NBER Program(s):Labor Studies, Technical Working Papers

We develop and analyze a tractable empirical model for strategic network formation that can be estimated with data from a single network at a single point in time. We model the network formation as a sequential process where in each period a single randomly selected pair of agents has the opportunity to form a link. Conditional on such an opportunity, a link will be formed if both agents view the link as beneficial to them. They base their decision on their own characateristics, the characteristics of the potential partner, and on features of the current state of the network, such as whether the two potential partners already have friends in common. A key assumption is that agents do not take into account possible future changes to the network. This assumption avoids complications with the presence of multiple equilibria, and also greatly simplifies the computational burden of anlyzing these models. We use Bayesian markov-chain-monte-carlo methods to obtain draws from the posterior distribution of interest. We apply our methods to a social network of 669 high school students, with, on average, 4.6 friends. We then use the model to evaluate the effect of an alternative assignment to classes on the topology of the network.

Motivation

Why do I want to teach you about Optimization?

Neural system ide

David A. Klindt, Alexander S.

(Submitted on 7 Nov 2

Neuroscientists cl
methods for neur
spaces that are sh
limitations: While
sample only a sm
networks (CNNs) t
surface thus far. V
Our network scale
ground-truth data
model outperform

Nichol

We
with
pro
Cor
The
feat
frie
net
sim
met
net
effe

t" and "where"

raman

Traditional
l feature
for data
that one can
ıeural
only at the
limensions.
cture on
ıetwork

ted
tial
nk.
em.
on
ave
the
atly
arlo
cial
the

Measuring and Bounding Experimenter Demand*

Jonathan de Quidt† Johannes Haushofer‡
Christopher Roth§

This version: May 26, 2017

Abstract

We propose a technique for assessing robustness of behavioral measures and treatment effects to experimenter demand effects. The premise is that by deliberately inducing demand in a structured way we can measure its influence and construct plausible bounds on demand-free behavior. We provide formal restrictions on choice that validate our method, and a Bayesian model that microfounds them. Seven pre-registered experiments with eleven canonical laboratory games and around 19,000 participants demonstrate the technique. We also illustrate how demand sensitivity varies by task, participant pool, gender, real versus hypothetical incentives, and participant attentiveness, and provide both reduced-form and structural analyses of demand effects.

Motivation

Why do I want to teach you about Optimization?

Neural system ide

David A. Klindt, Alexander S.

(Submitted on 7 Nov 2

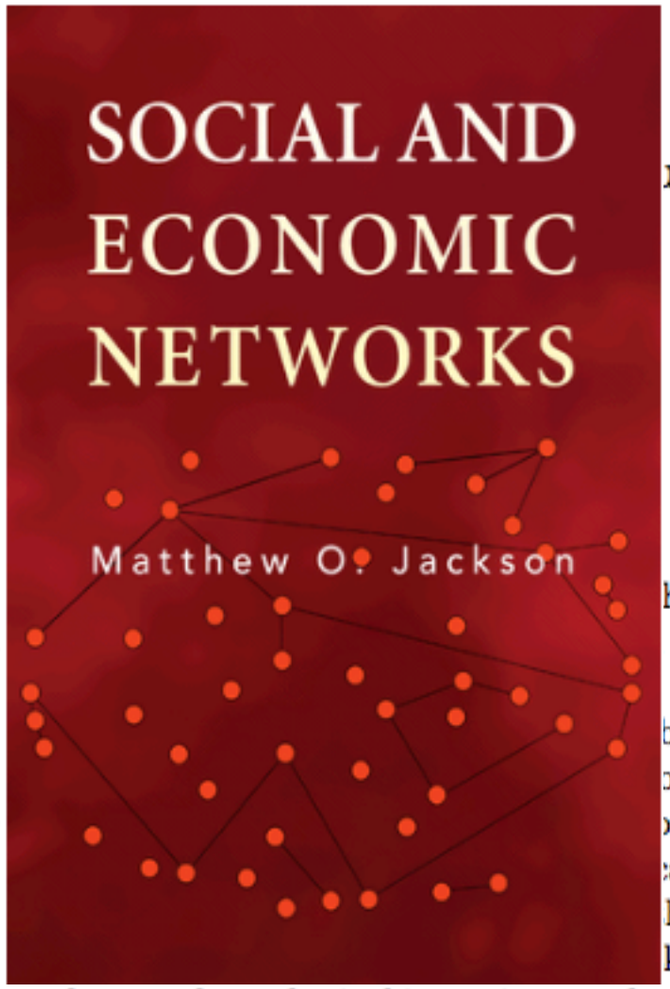
Neuroscientists cl
methods for neura
spaces that are sh
limitations: While
sample only a sma
networks (CNNs) t
surface thus far. V
Our network scale
ground-truth data
model outperform

Nichol

We
with
pro
Cor
The
feat
frie
net
sim
met
net
effe

Measuring and Bounding Experimenter

Jonat ushofer‡



We propose havioral mea-
sures and t effects. The
premise is t a structured
way we can ble bounds on
demand-free on choice that
validate our ofounds them.
Seven pre-re al laboratory
games and a he technique.
We also illus k, participant
pool, gender, real versus hypothetical incentives, and participant attentiveness, and provide both reduced-form and structural analyses of demand effects.

t" and "where"

raman

Traditional
l feature
for data
that one can
neural
only at the
limensions.
cture on
network

ted
tial
nk.
em.
on
ave
the
atly
arlo
cial
the

Motivation

Why do I want to teach you about Optimization?

Neural system ide

David A. Klindt, Alexander S.

(Submitted on 7 Nov 2

Neuroscientists cl
methods for
spaces that a
limitations: V
sample only
networks (CN
surface thus
Our network
ground-truth data
model outperform

Measuring and Bounding Experimenter

SOCIAL AND

Jonat ushofer‡

Matthew O. Jackson

We propose
sures and ti
premise is t
way we can
demand-free
validate our
Seven pre-re
games and
We also illus

havioral mea-
effects. The
a structured
ble bounds on
on choice that
ofounds them.
al laboratory
he technique.
k, participant

pool, gender, real versus hypothetical incentives, and participant attentiveness, and provide both reduced-form and structural analyses of demand effects.

t" and "where"

Traditional
l feature
for data
that one can
neural
only at the
limensions.
cture on
network

nk.
em.
on
ave
the
atly
arlo
cial
the

You'll eventually (or maybe even now) need to build you own estimators (models) that are not yet in Scikit-Learn.

Motivation

Learn how to

1) Build (high-dimensional) models of/for neural & behavioral data

$$\text{output} = F_{[\text{params}]}(\text{input})$$

*formulate **F** mathematically on on a computer*

2) Train such models

$$\text{data\_slice\_2} = F_{[\text{params}]}(\text{data\_slice\_1})$$

*determine **params** from pair of linked data tensors*

3) Evaluate and compare such models

$$\text{new\_data\_slice\_2} = F_{[\text{params}]}(\text{new\_data\_slice\_1})$$

these two you've already been seeing for several weeks

Motivation

Learn how to

1) Build (high-dimensional) models of/for neural & behavioral data

$$\text{output} = F_{[\text{params}]}(\text{input})$$

*formulate **F** mathematically on on a computer*

2) Train such models

data_sl ...ine ***params*** *from pair of linked data tensors*

This part was somehow taken care of for you by the package

3) Evaluate and compare such models

$$\text{new\_data\_slice\_2} = F_{[\text{params}]}(\text{new\_data\_slice\_1})$$

these two you've already been seeing for several weeks

Motivation

Learn how to

1) Build (high-dimensional) models of/for neural & behavioral data

This part was somehow taken care of for you by the package

(input)

*formulate **F** mathematically on on a computer*

except in certain special cases:

1. distance classifier
2. OLS regressor
3. k-means clustering
4. PCA

2) Train such m

*ne **params***

air of linked

a tensors

3) Evaluate an

new_data_slice_2 = $F_{[params]}$(new_data_slice_1)

these two you've already been seeing for several weeks

Motivation

Learn how to

1) Build (high-dimensional) models of/for neural & behavioral data

This part was somehow taken care of for you by the package

(input)

*formulate **F** mathematically on on a computer*

except in certain special cases:

1. distance classifier (but **not** SVM)
2. OLS regressor (but **not** Lasso regression)
3. k-means clustering (but **not** GMM)
4. PCA (but **not** Factor Analysis)

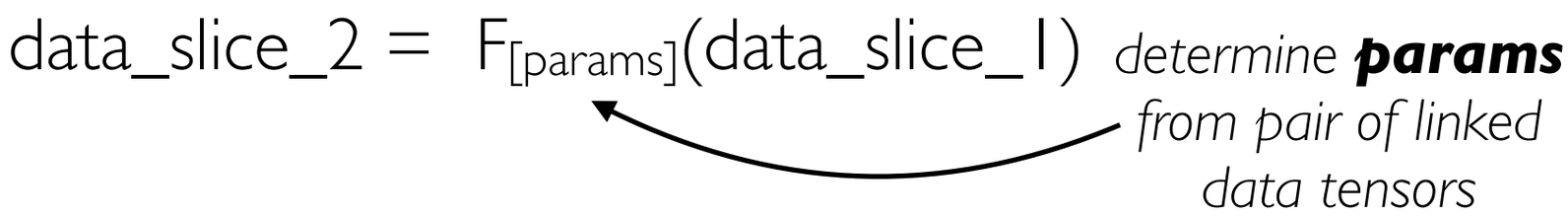
2) Train

3) Evalu

new_data_slice_2 = f[params](new_data_slice_1)

these two you've already been seeing for several weeks

Motivation



You'll eventually (or maybe even now) need to build you own estimators (models) that are not yet in Scikit-Learn.

But how?

Motivation

data_slice_2 = $F_{[params]}$(data_slice_1)

*determine **params** from pair of linked data tensors*

You'll eventually (or maybe even now) need to build you own estimators (models) that are not yet in Scikit-Learn.

But how?

Key fact: Amazingly enough, there's a generic "one-size-fits-all" method— if sometimes suboptimal — for building and training models.

Motivation

Goal: Find F and parameters $\boldsymbol{\Theta}$ such that:

$$\mathrm{data}_2 = F_\theta(\mathrm{data}_1)$$

determine ***params*** *from pair of linked data tensors*

Motivation

Goal: Find F and parameters $\boldsymbol{\Theta}$ such that:

$$\mathrm{data}_2 = F_\theta(\mathrm{data}_1)$$

determine ***params*** *from pair of linked data tensors*

Loss Function:

$$\mathcal{L}(\mathrm{data}_2, F_\theta(\mathrm{data}_1))$$

such that

$$\mathcal{L}(x, y) = 0 \quad \Leftrightarrow \quad x \cong y$$

Motivation

Goal: Find F and parameters $\boldsymbol{\Theta}$ such that:

$$\mathrm{data}_2 = F_\theta(\mathrm{data}_1)$$

determine ***params*** *from pair of linked data tensors*

Loss Function:

$$\mathcal{L}(\mathrm{data}_2, F_\theta(\mathrm{data}_1))$$

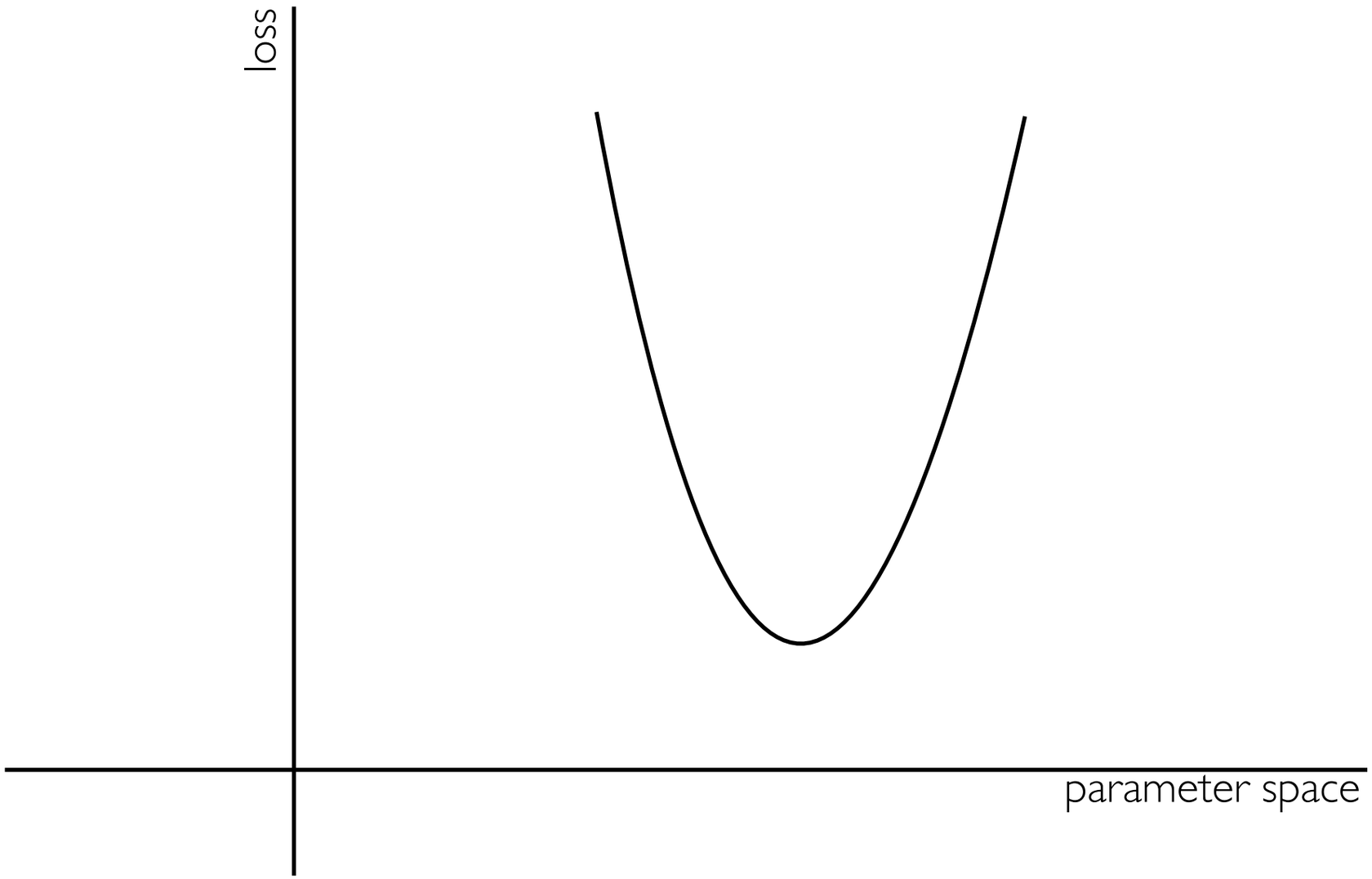
such that

$$\mathcal{L}(x, y) = 0 \quad \Leftrightarrow \quad x \cong y$$

For example (but not limited to):

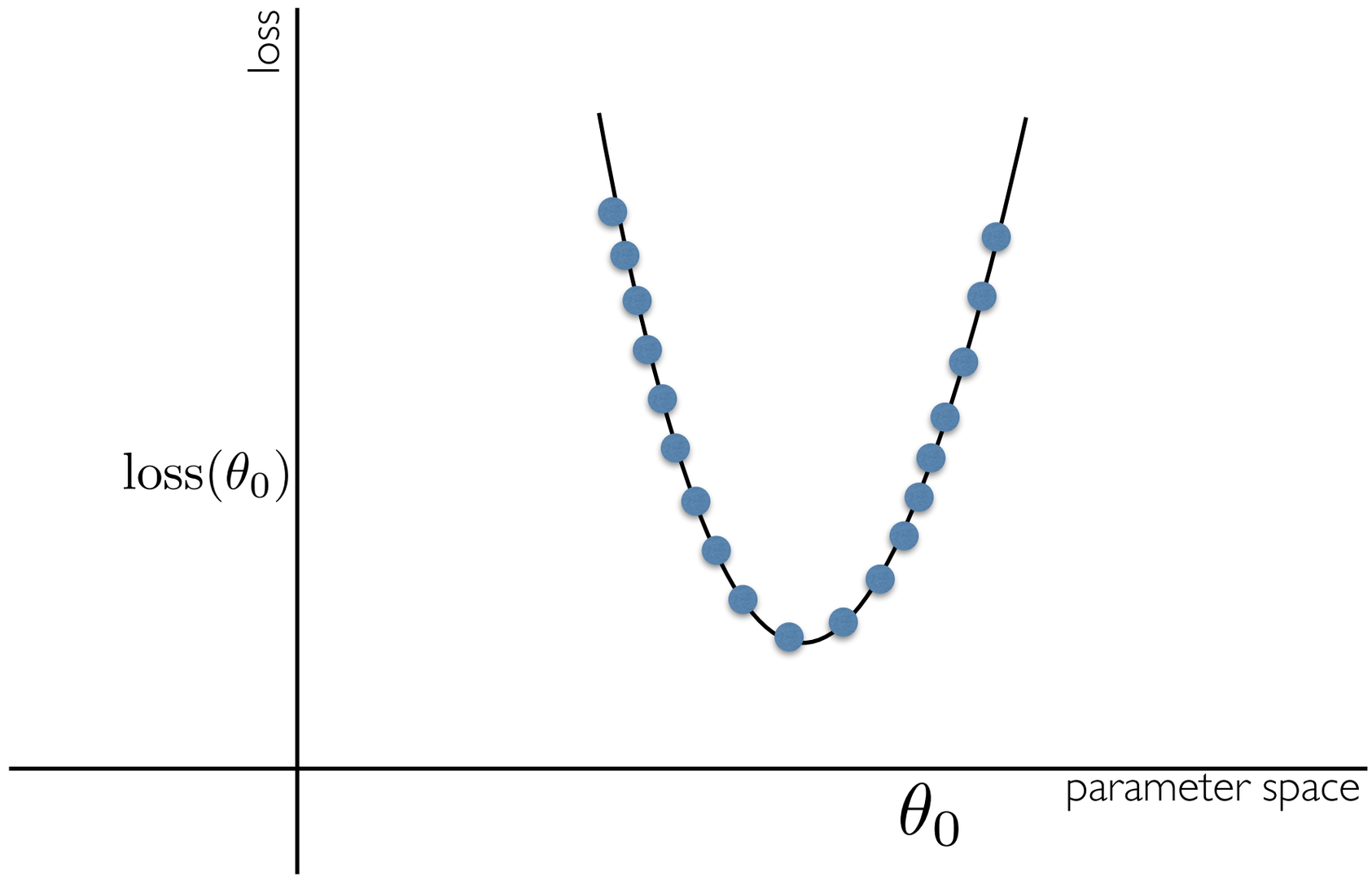
$$\mathcal{L}(\mathrm{data}_2, F_\theta(\mathrm{data}_1)) = ||\mathrm{data}_2 - F_\theta(\mathrm{data}_1)||_2^2$$

Optimization



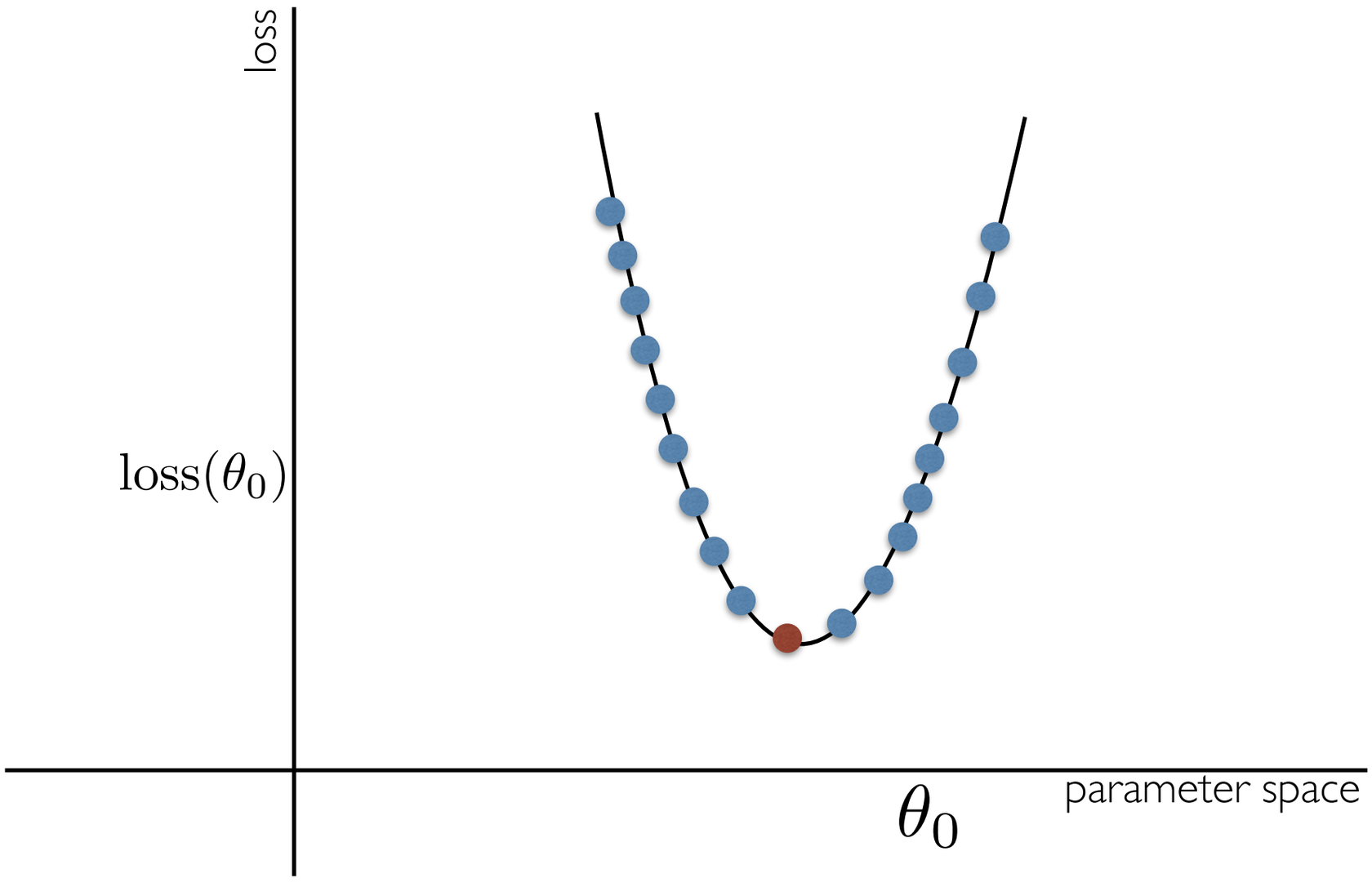
Optimization

option 1: try many options . . .



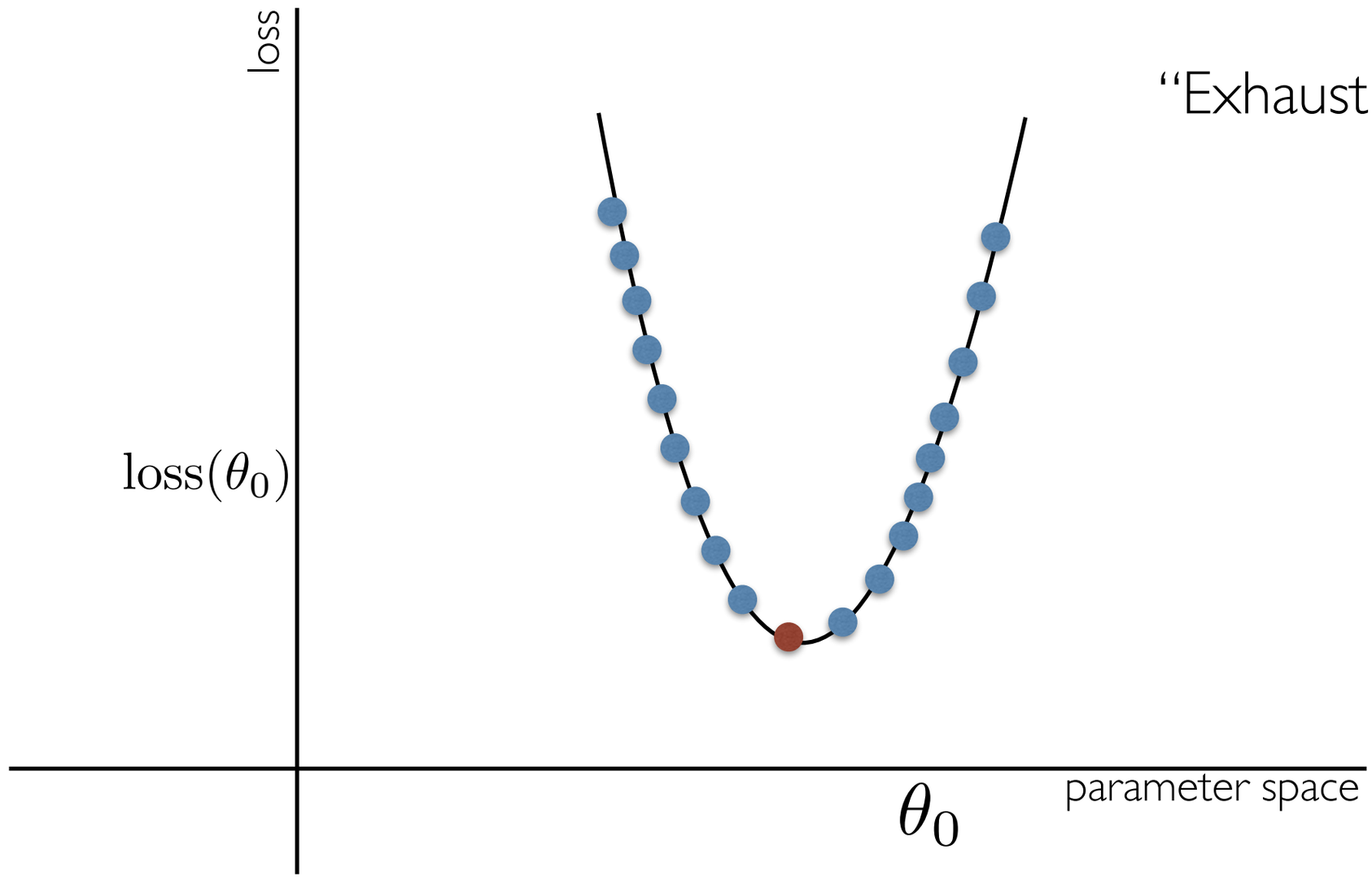
Optimization

option 1: try many options . . . pick **the best**



Optimization

option 1: try many options . . . pick **the best**



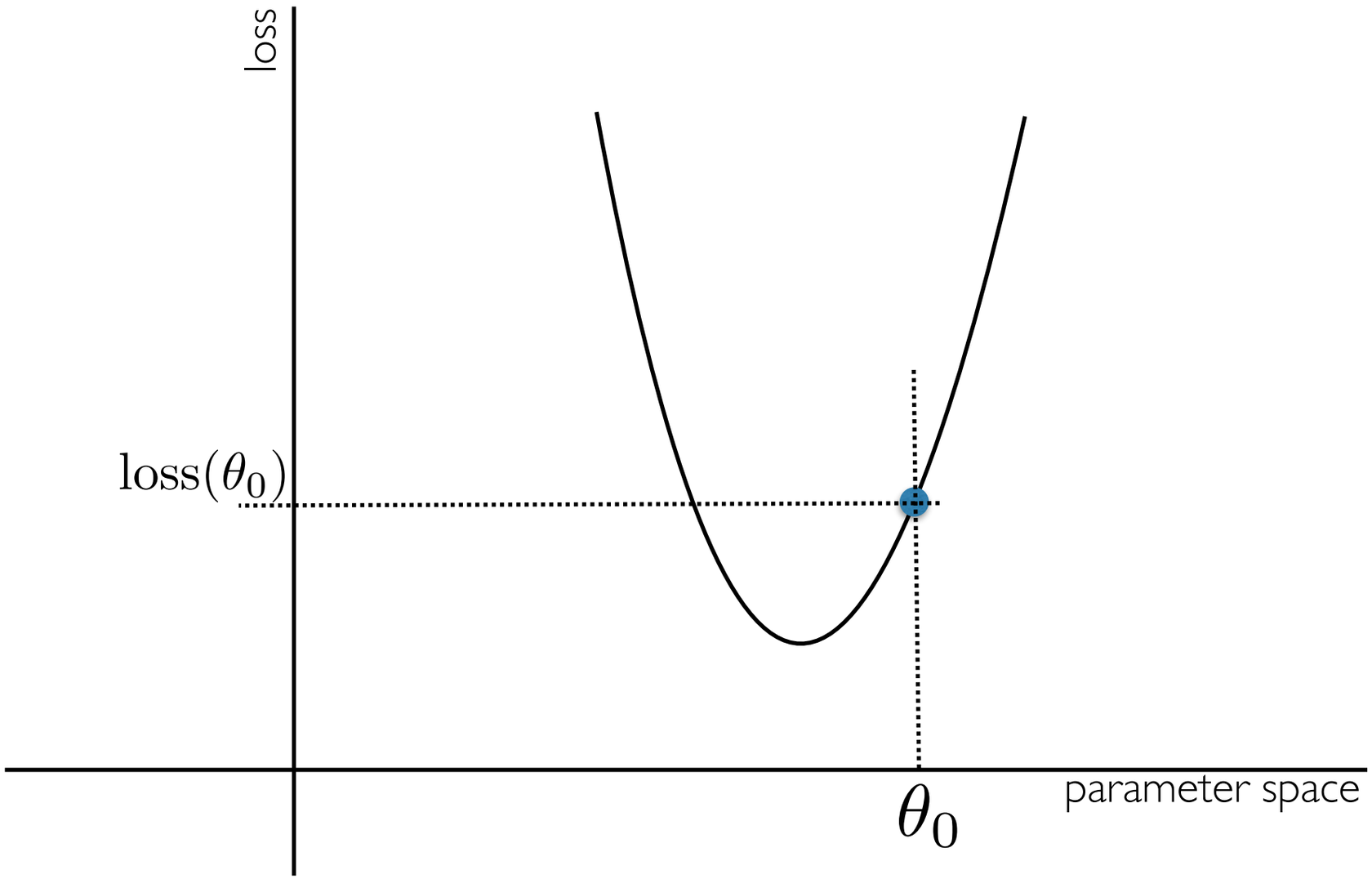
"Exhaustive" or "grid" search

pros: guaranteed to eventually find best minimum; numerically stable

con: infeasible. exhausting. (takes forever)

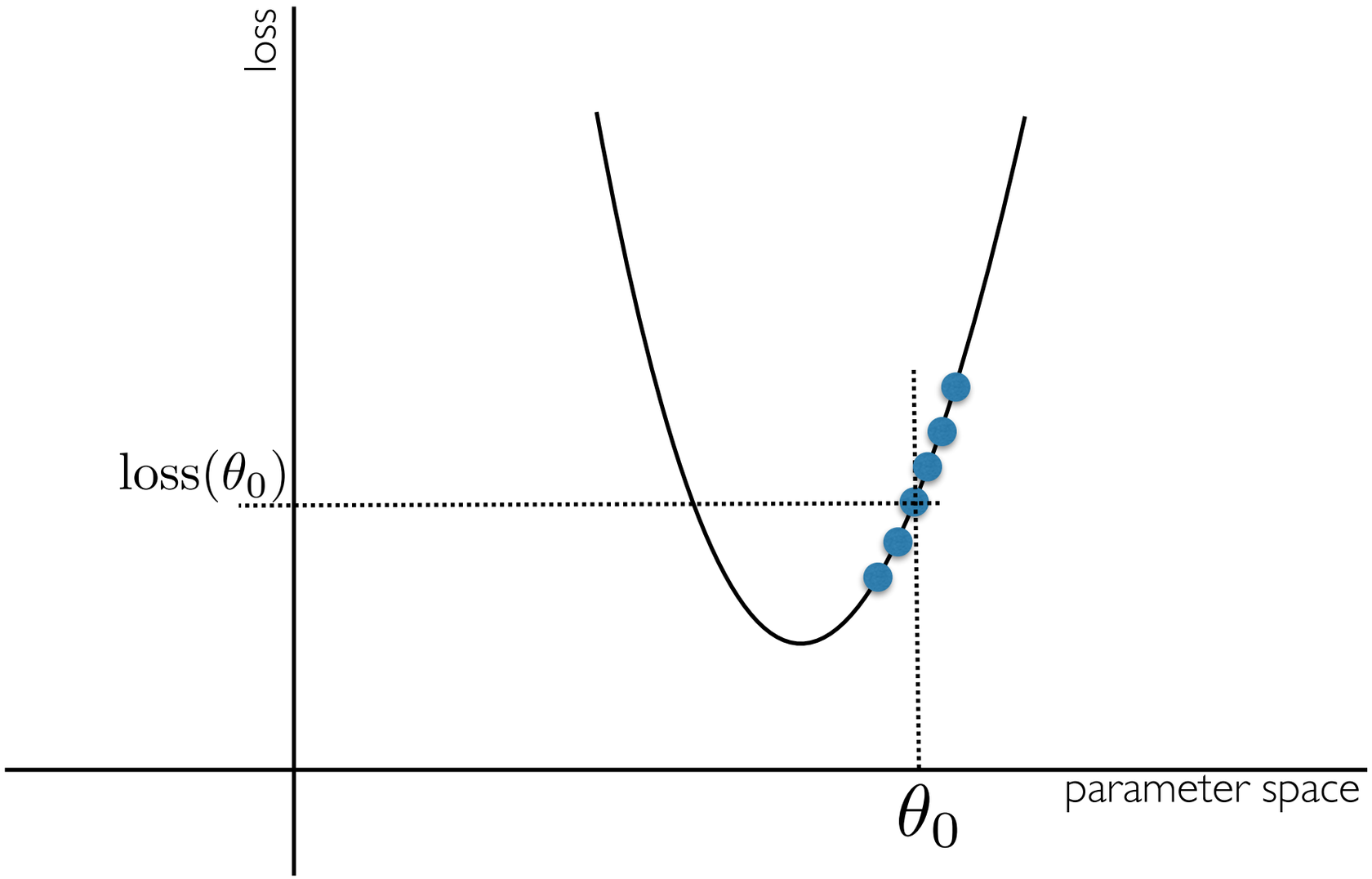
Optimization

option 2: start somewhere



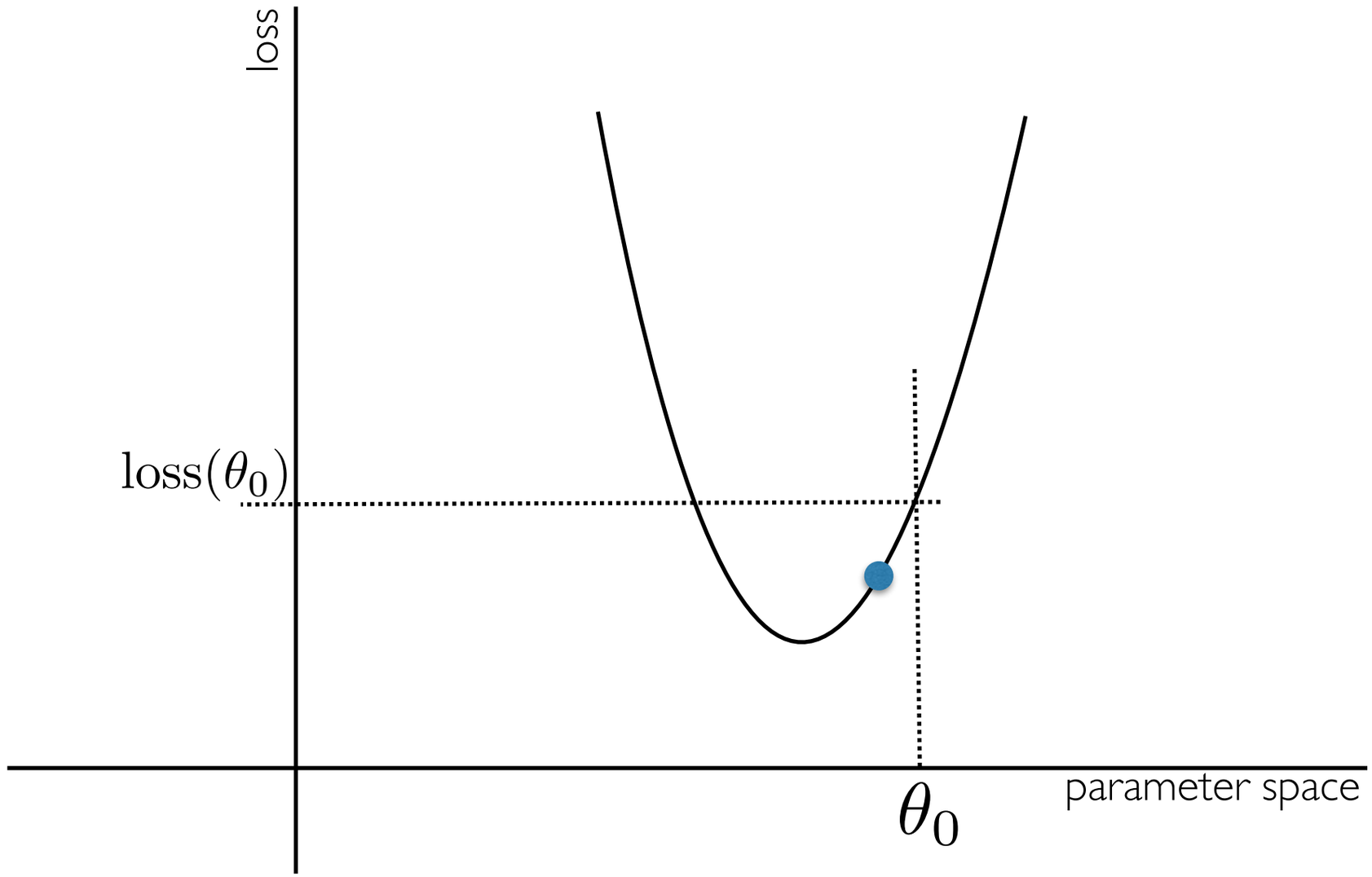
Optimization

option 2: start somewhere, try nearby options



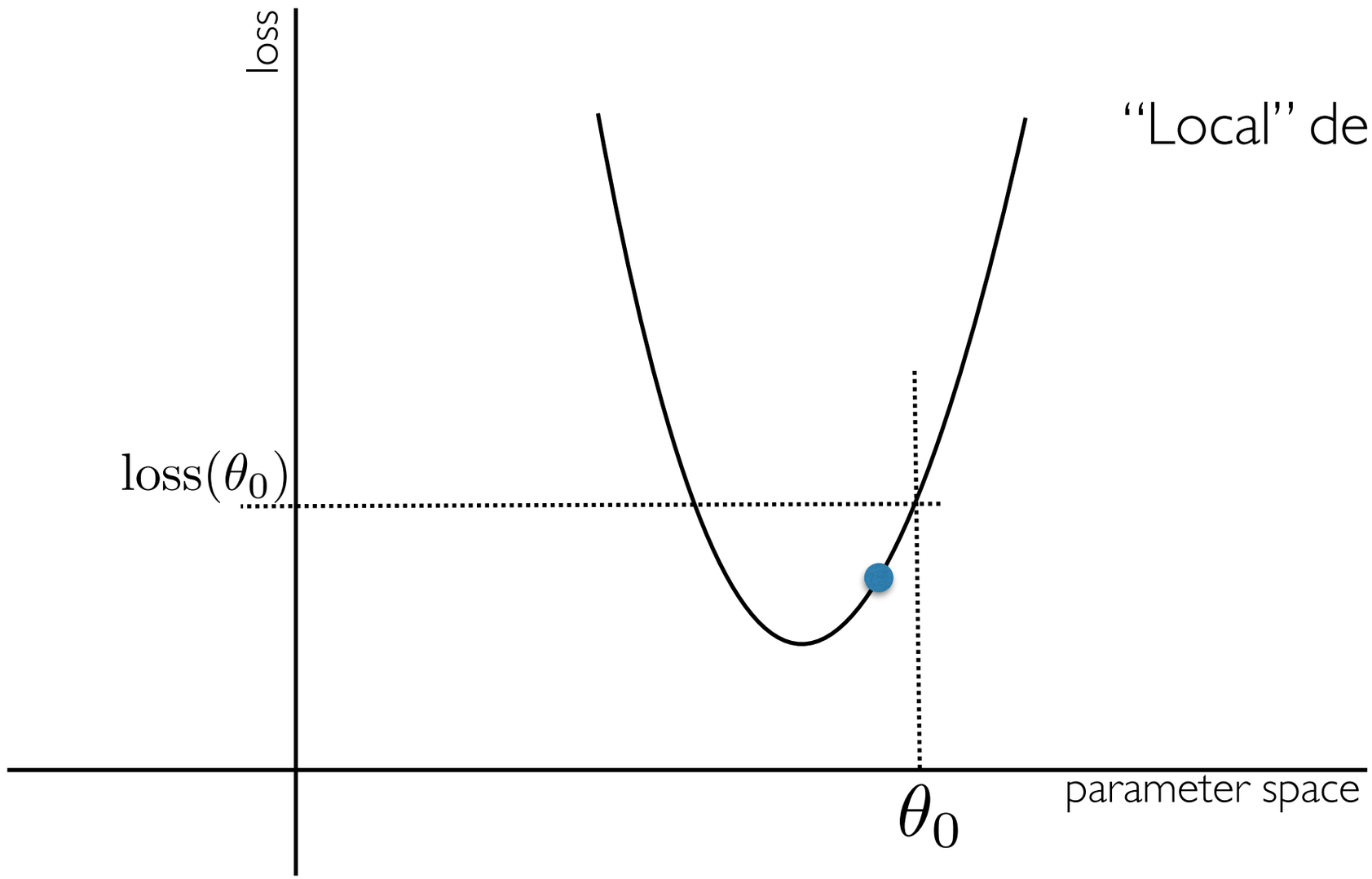
Optimization

option 2: start somewhere, try nearby options, pick the best, iterate



Optimization

option 2: start somewhere, try nearby options, pick the best, iterate



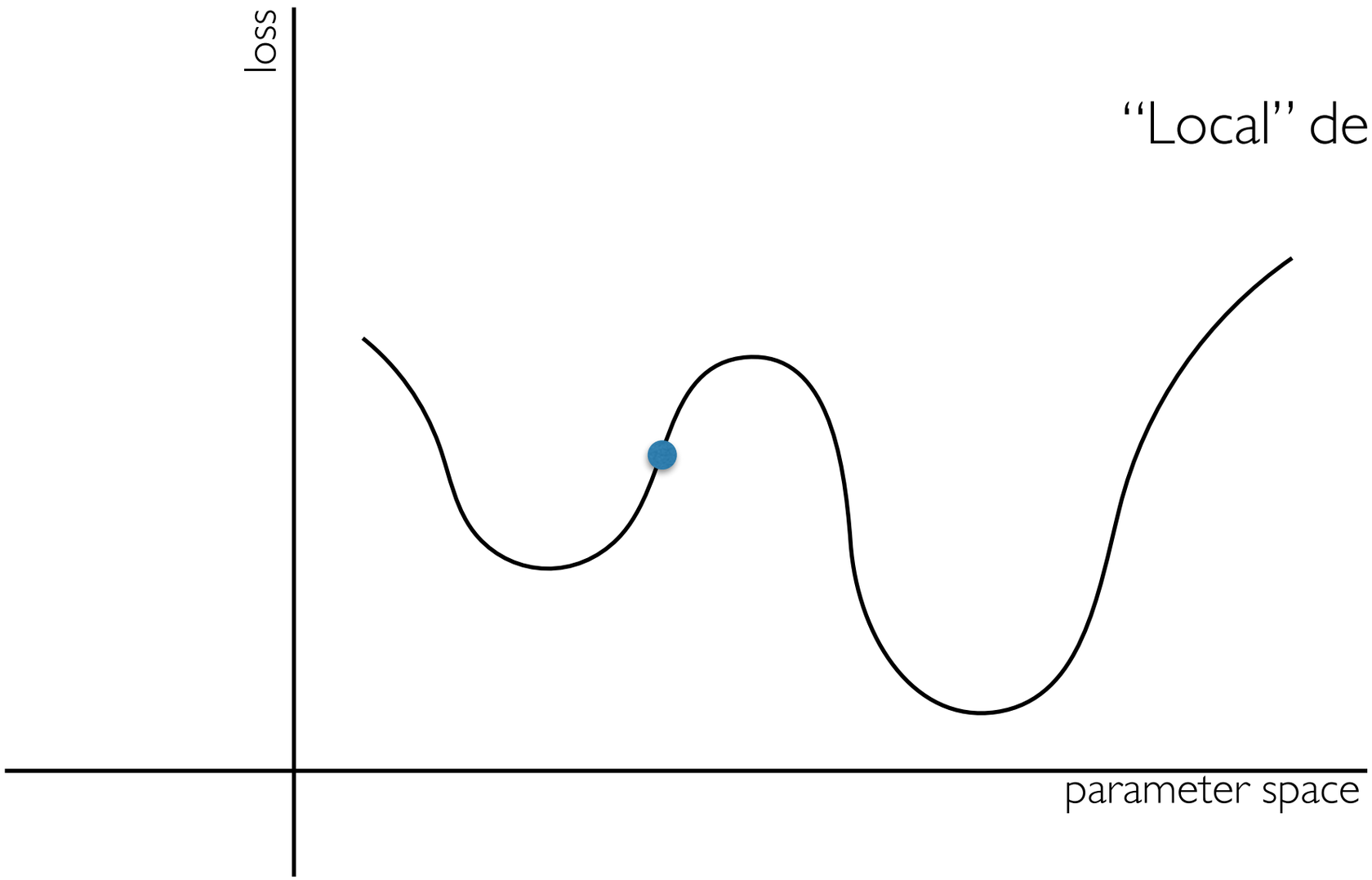
"Local" derivative-free methods

pro: numerically stable

con: not guaranteed to work, still potentially inefficient

Optimization

option 2: start somewhere, try nearby options, pick the best, iterate

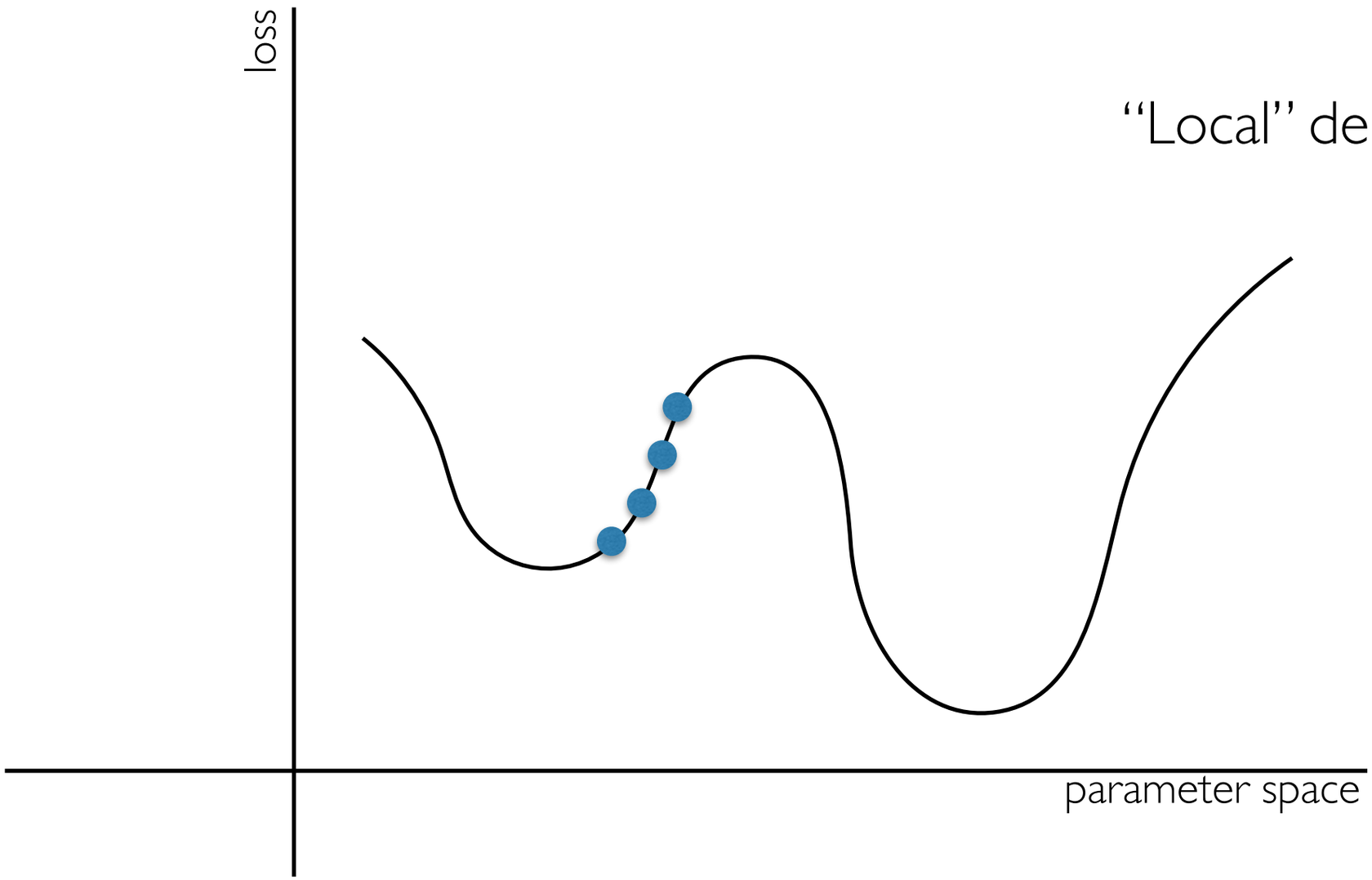


"Local" derivative-free methods

con: not guaranteed to work because of local minima

Optimization

option 2: start somewhere, try nearby options, pick the best, iterate

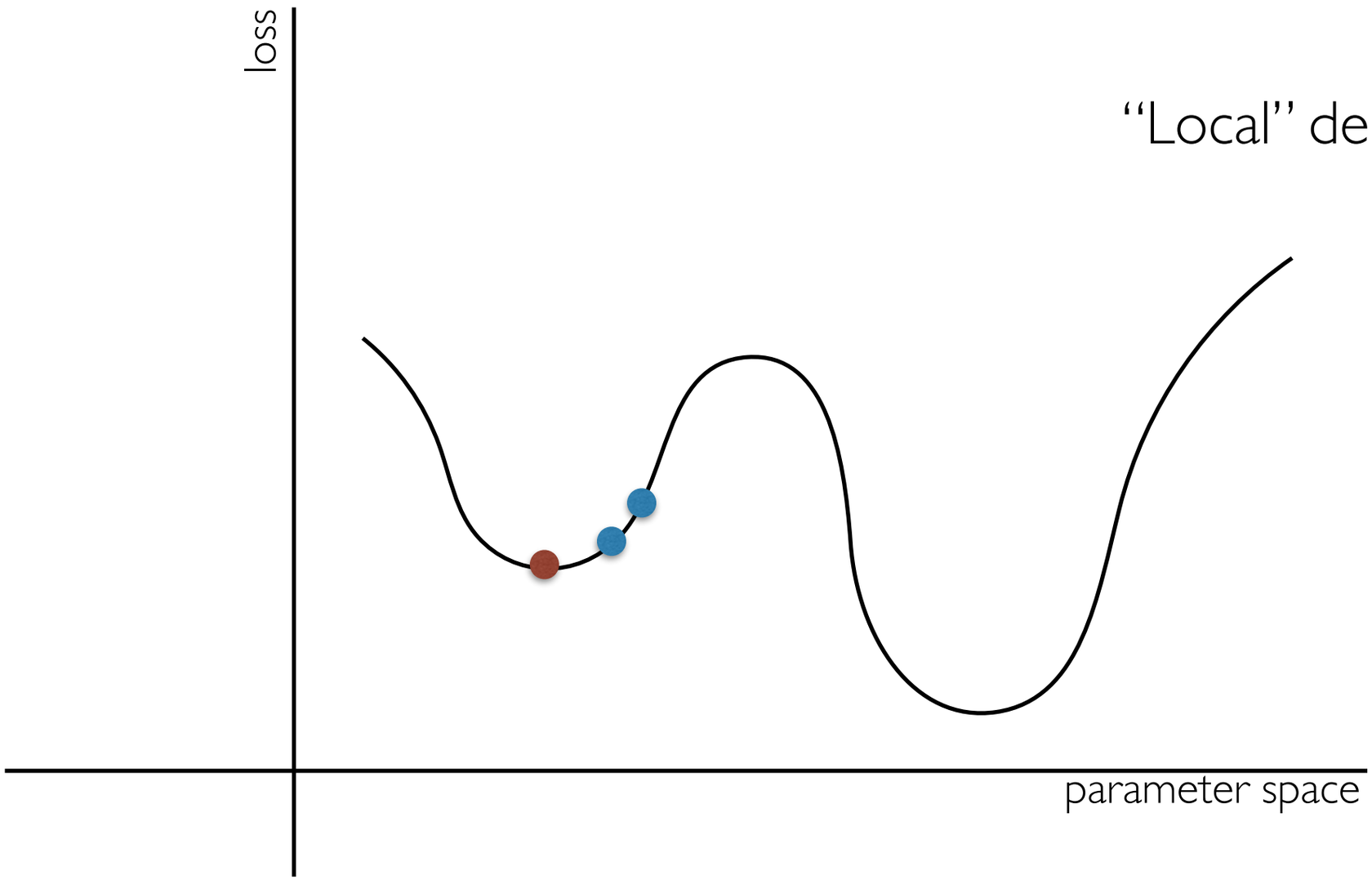


"Local" derivative-free methods

con: not guaranteed to work because of local minima

Optimization

option 2: start somewhere, try nearby options, pick the best, iterate

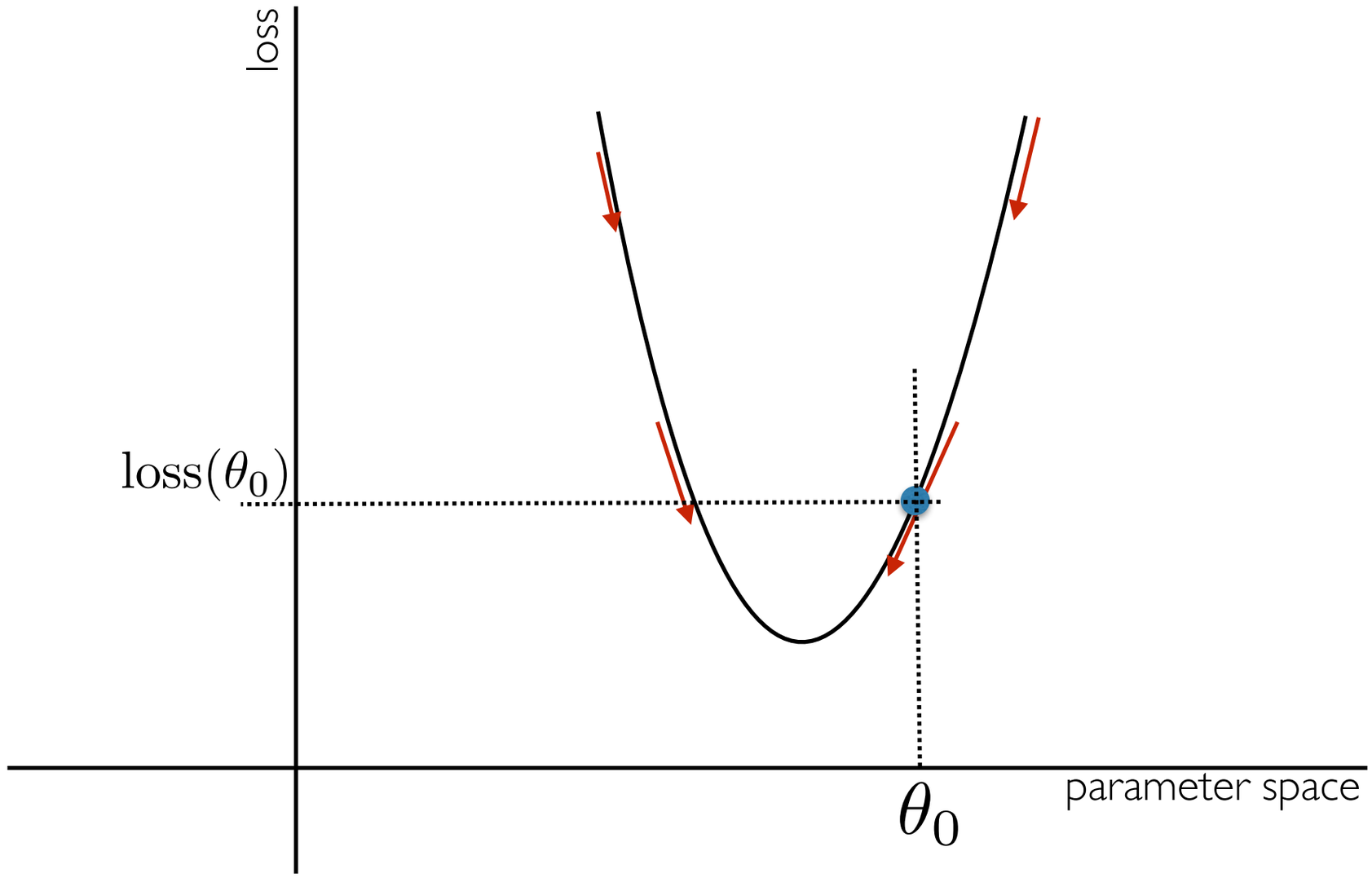


"Local" derivative-free methods

con: not guaranteed to work because of local minima

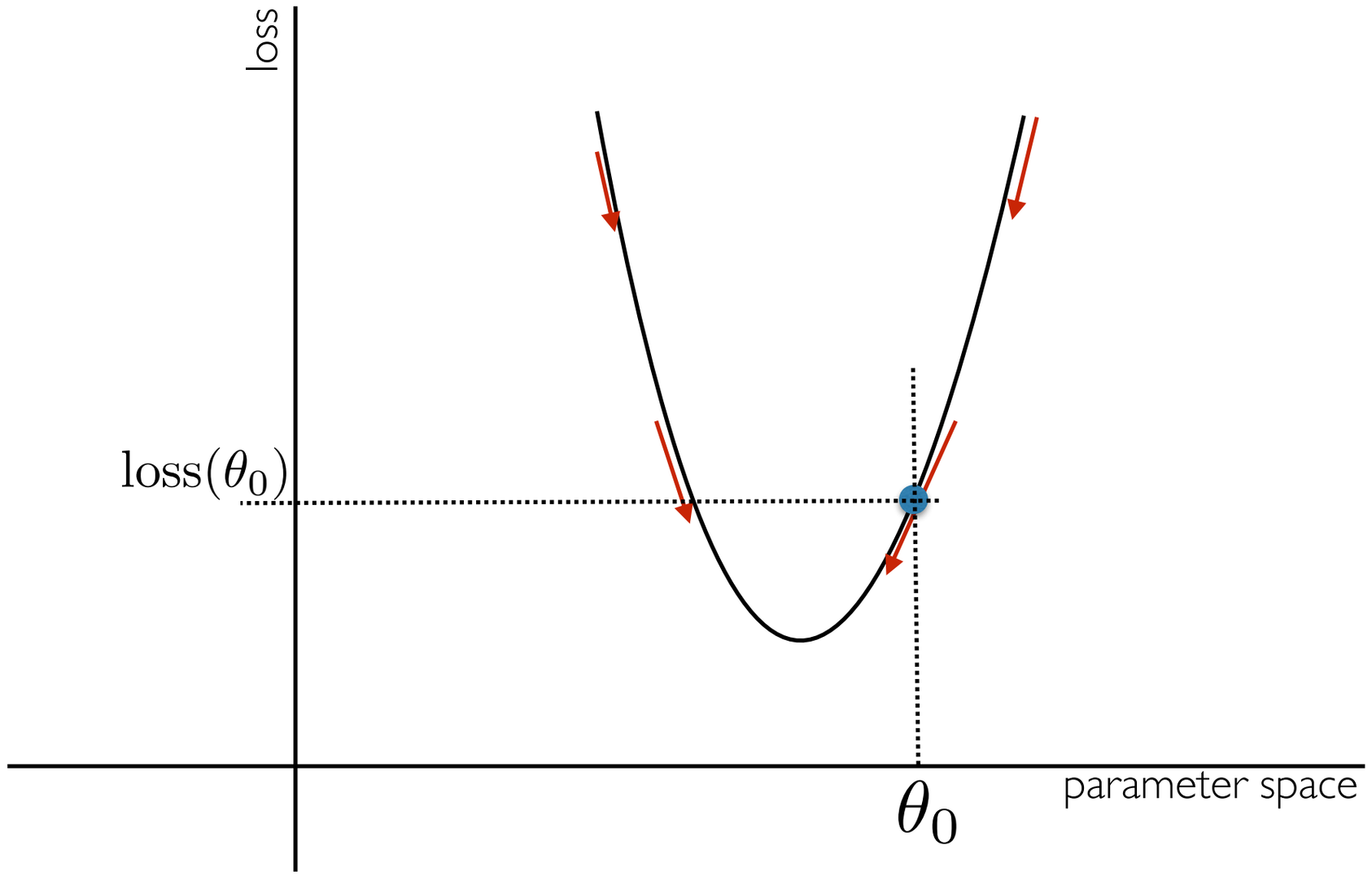
Optimization

option 3: follow the curve



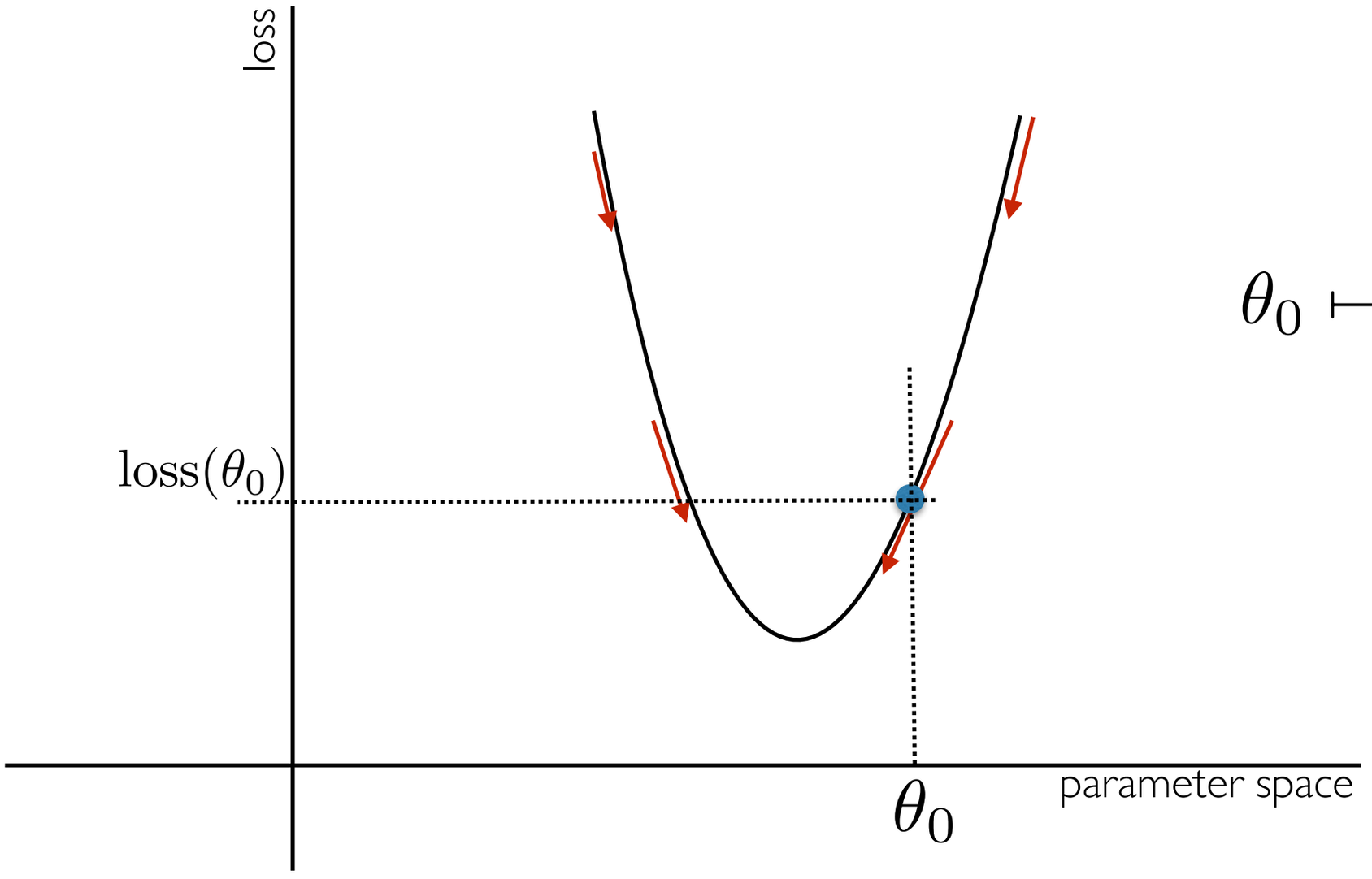
Optimization

option 3: follow the (negative of) the loss gradient



Optimization

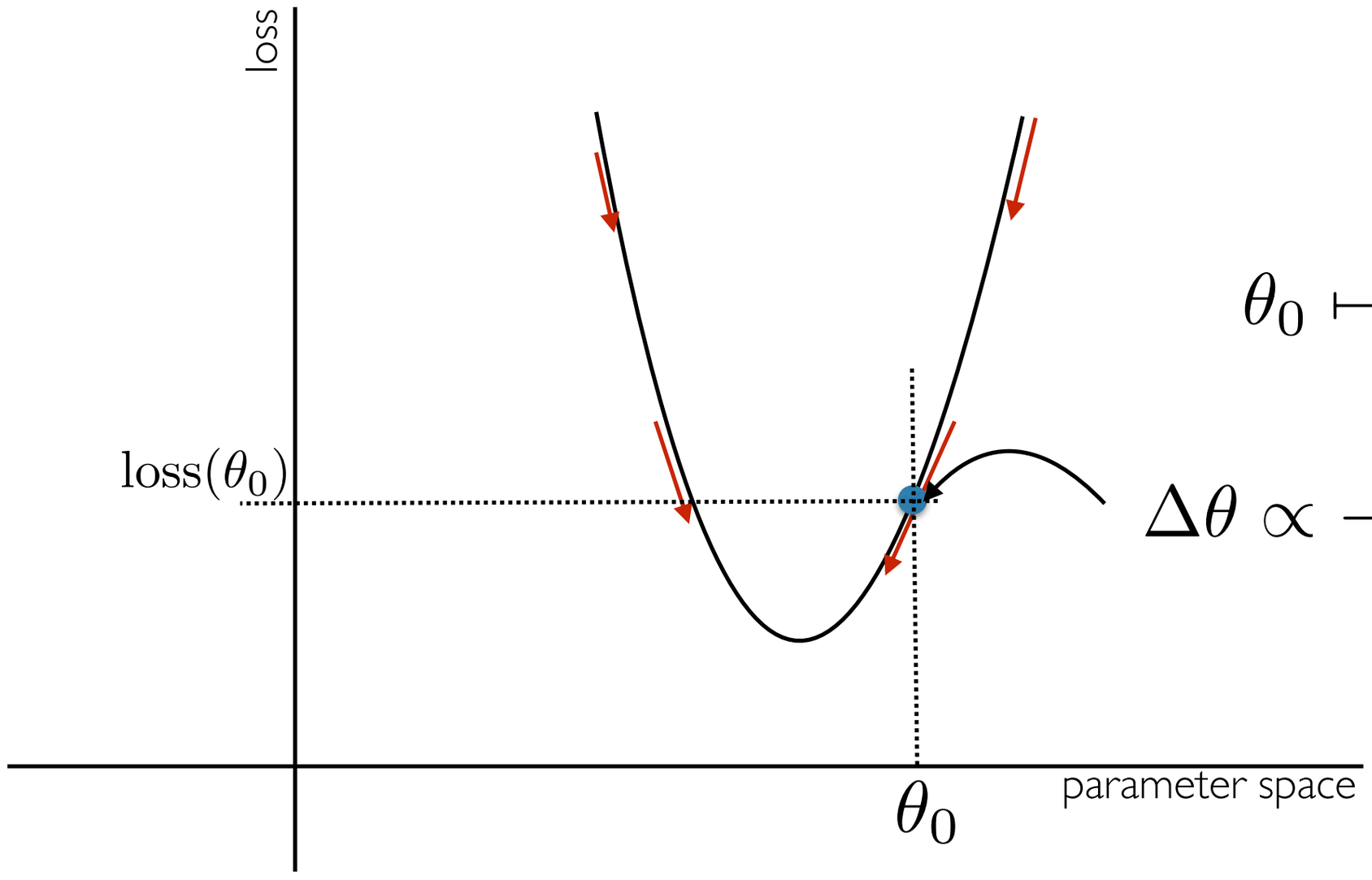
option 3: follow the (negative of) the loss gradient



$$\theta_0 \mapsto \theta_0 + \Delta\theta$$

Optimization

option 3: follow the (negative of) the loss gradient

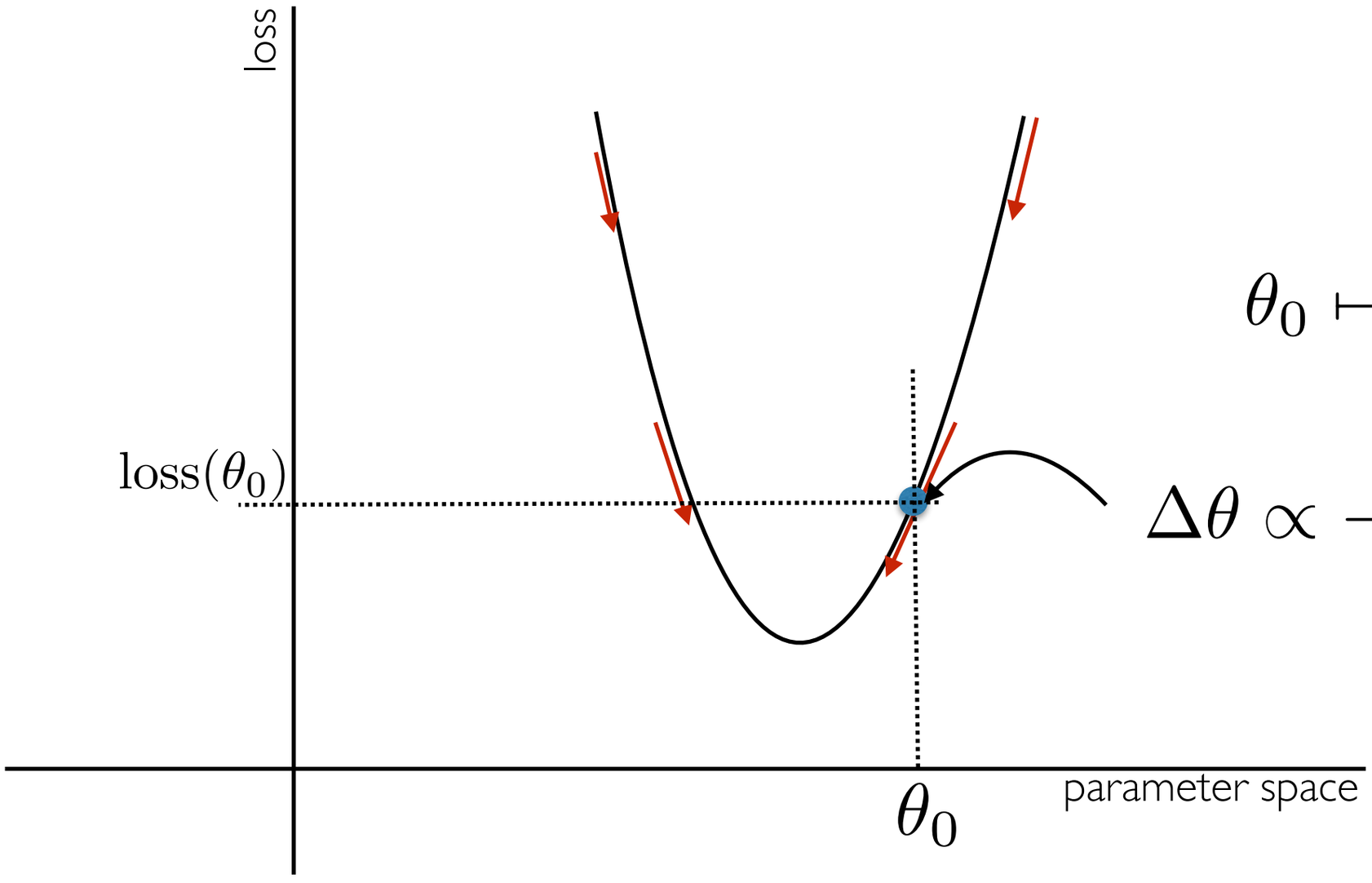


$$\theta_0 \mapsto \theta_0 + \Delta\theta$$

$$\Delta\theta \propto - \left. \frac{\partial[\text{loss}(\theta)]}{\partial\theta} \right|_{\theta=\theta_0}$$

Optimization

option 3: follow the (negative of) the loss gradient



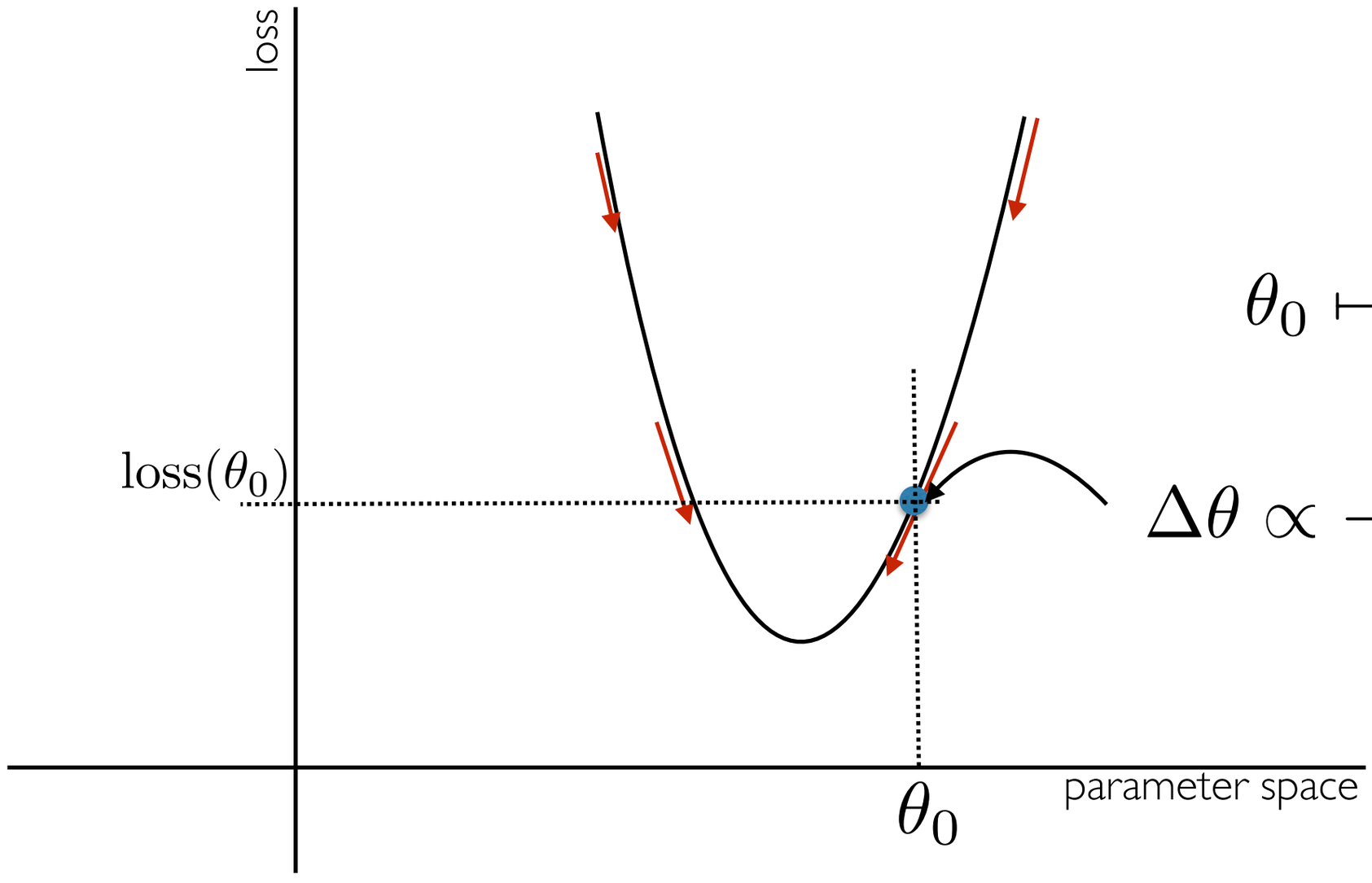
$$\theta_0 \mapsto \theta_0 + \Delta\theta$$

$$\Delta\theta \propto -\left.\frac{\partial[\text{loss}(\theta)]}{\partial\theta}\right|_{\theta=\theta_0}$$

pros: efficient

Optimization

option 3: follow the (negative of) the loss gradient

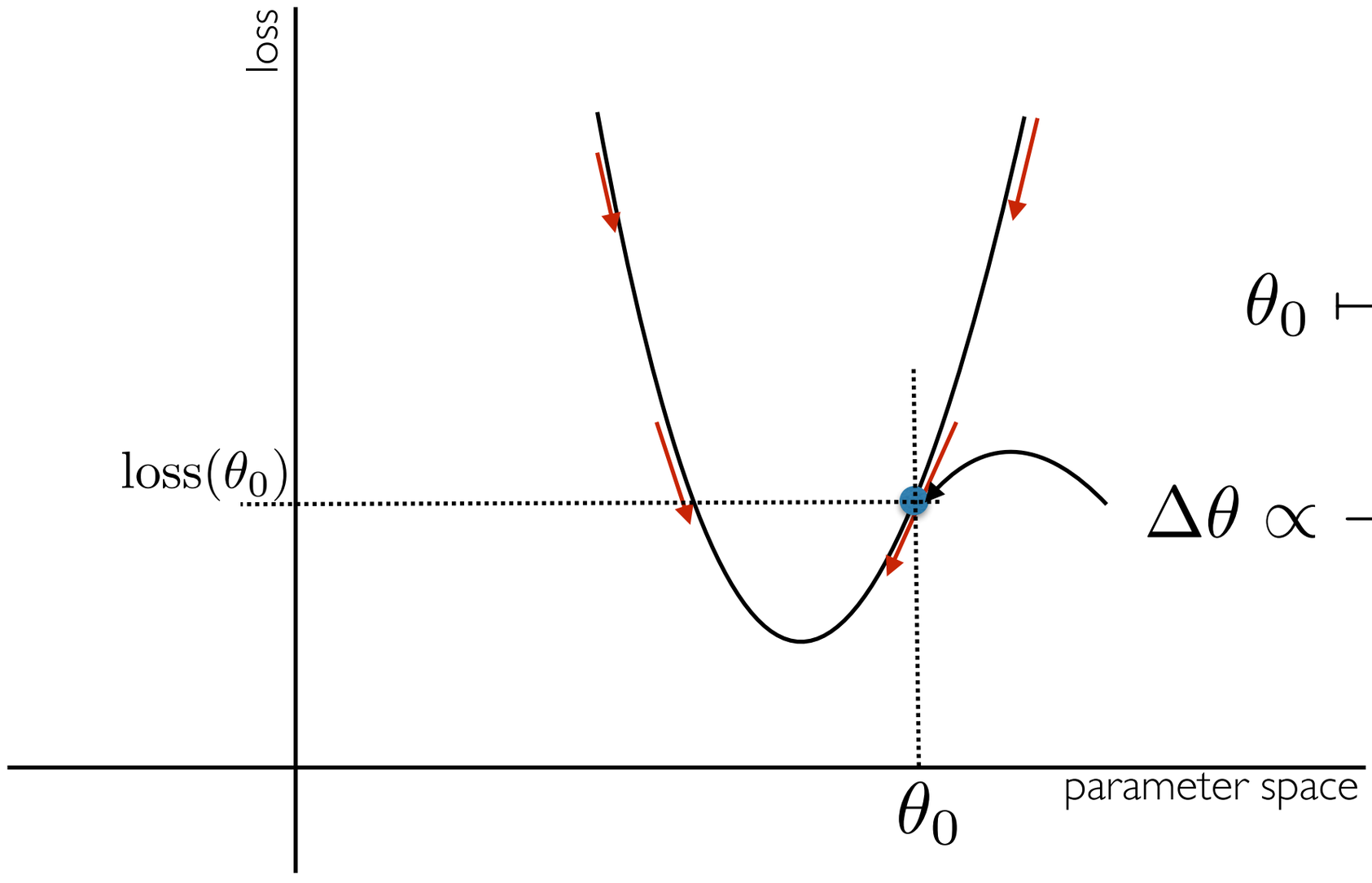


pros: efficient

con: local minima problem; not necessarily numerically stable

Optimization

option 3: follow the (negative of) the loss gradient



$$\theta_0 \mapsto \theta_0 + \Delta\theta$$

$$\Delta\theta \propto -\left.\frac{\partial[\text{loss}(\theta)]}{\partial\theta}\right|_{\theta=\theta_0}$$

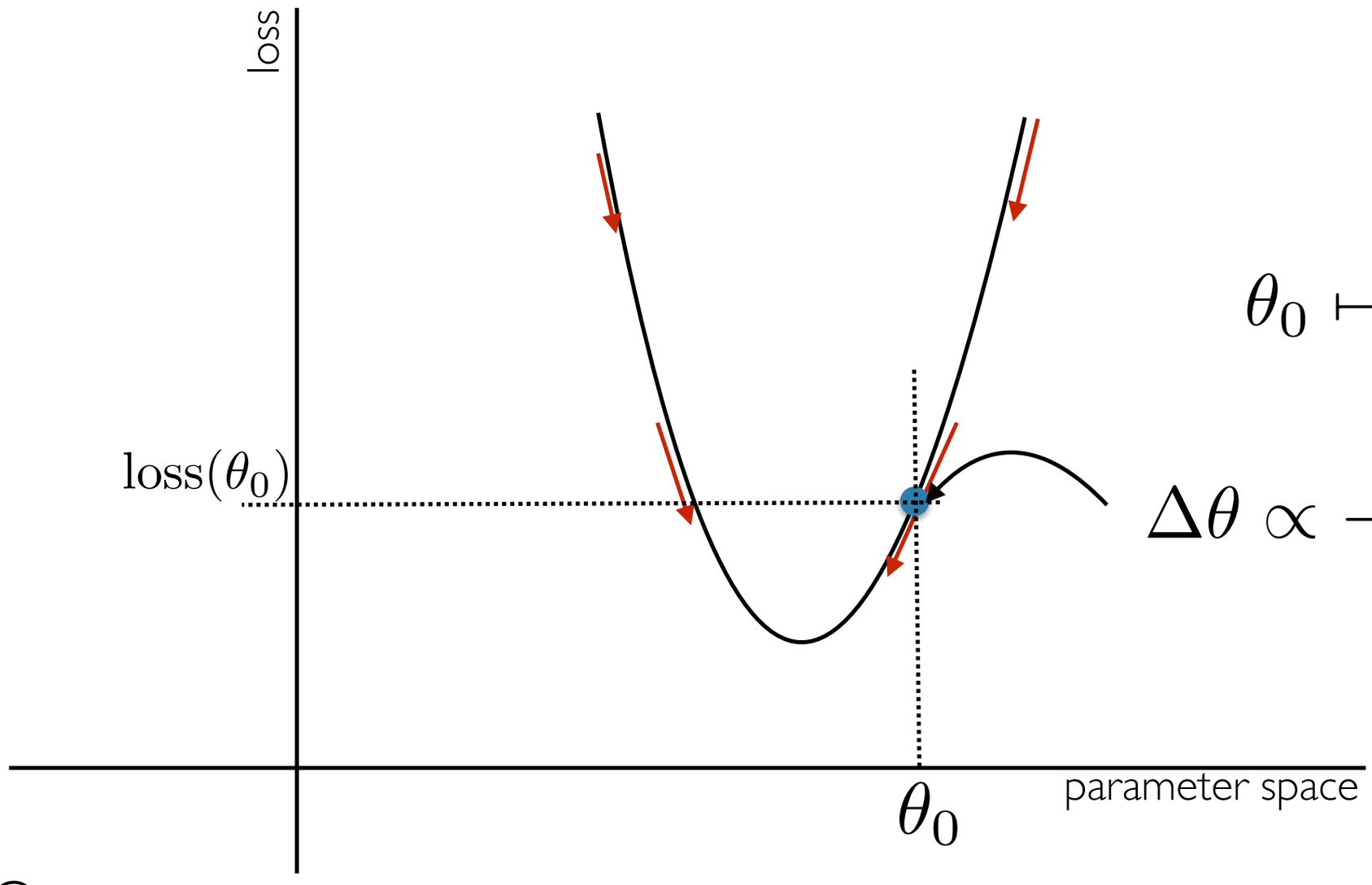
pros: efficient

con: local minima problem; not necessarily numerically stable

we'll see more on this topic shortly

Optimization

option 3: follow the (negative of) the loss gradient



$$\theta_0 \mapsto \theta_0 + \Delta\theta$$

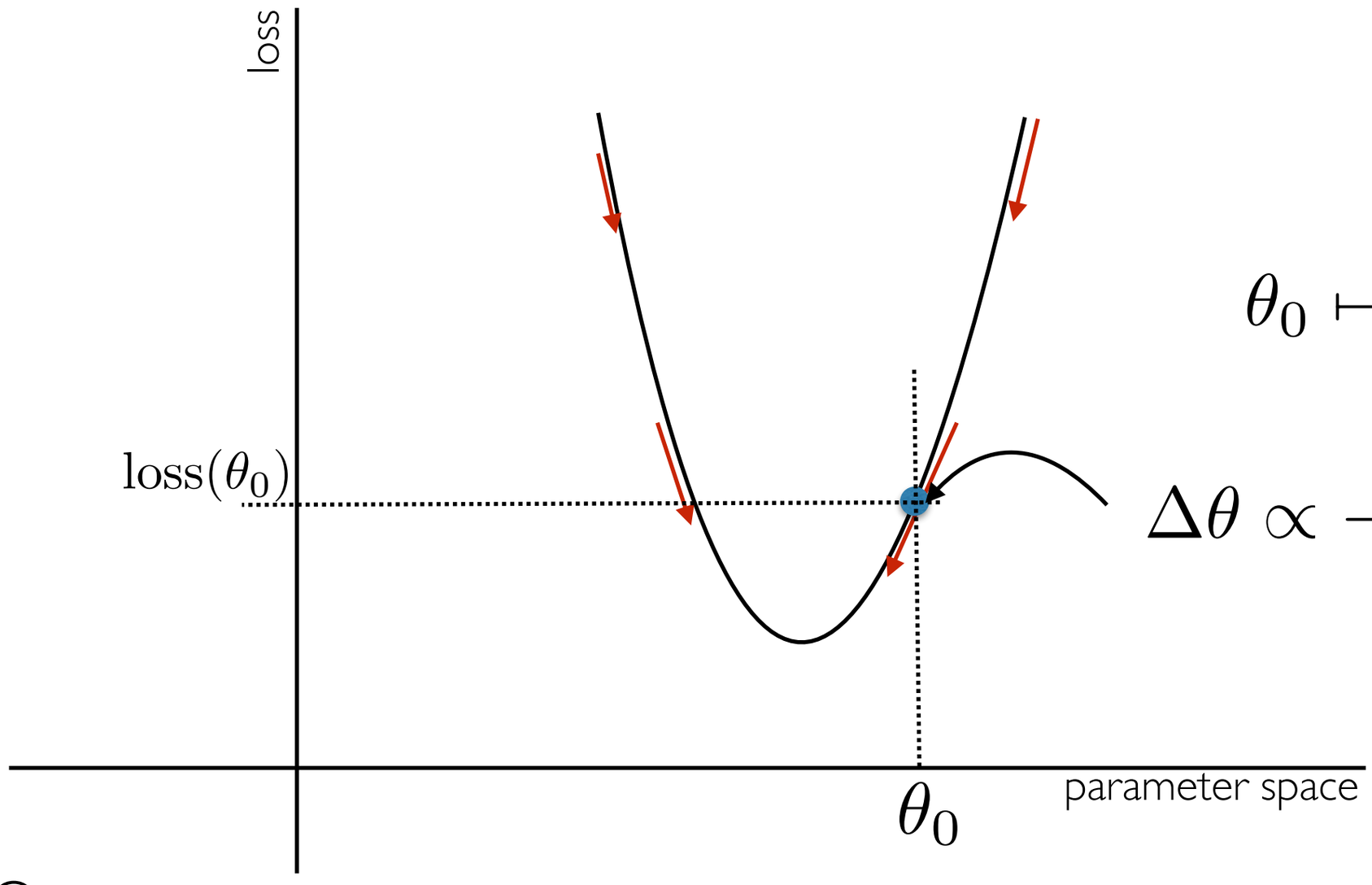
$$\Delta\theta \propto -\left.\frac{\partial[\text{loss}(\theta)]}{\partial\theta}\right|_{\theta=\theta_0}$$

ALSO:

1) loss needs to be differentiable (or close to it) in parameters

Optimization

option 3: follow the (negative of) the loss gradient



$$\theta_0 \mapsto \theta_0 + \Delta\theta$$

$$\Delta\theta \propto -\left.\frac{\partial[\text{loss}(\theta)]}{\partial\theta}\right|_{\theta=\theta_0}$$

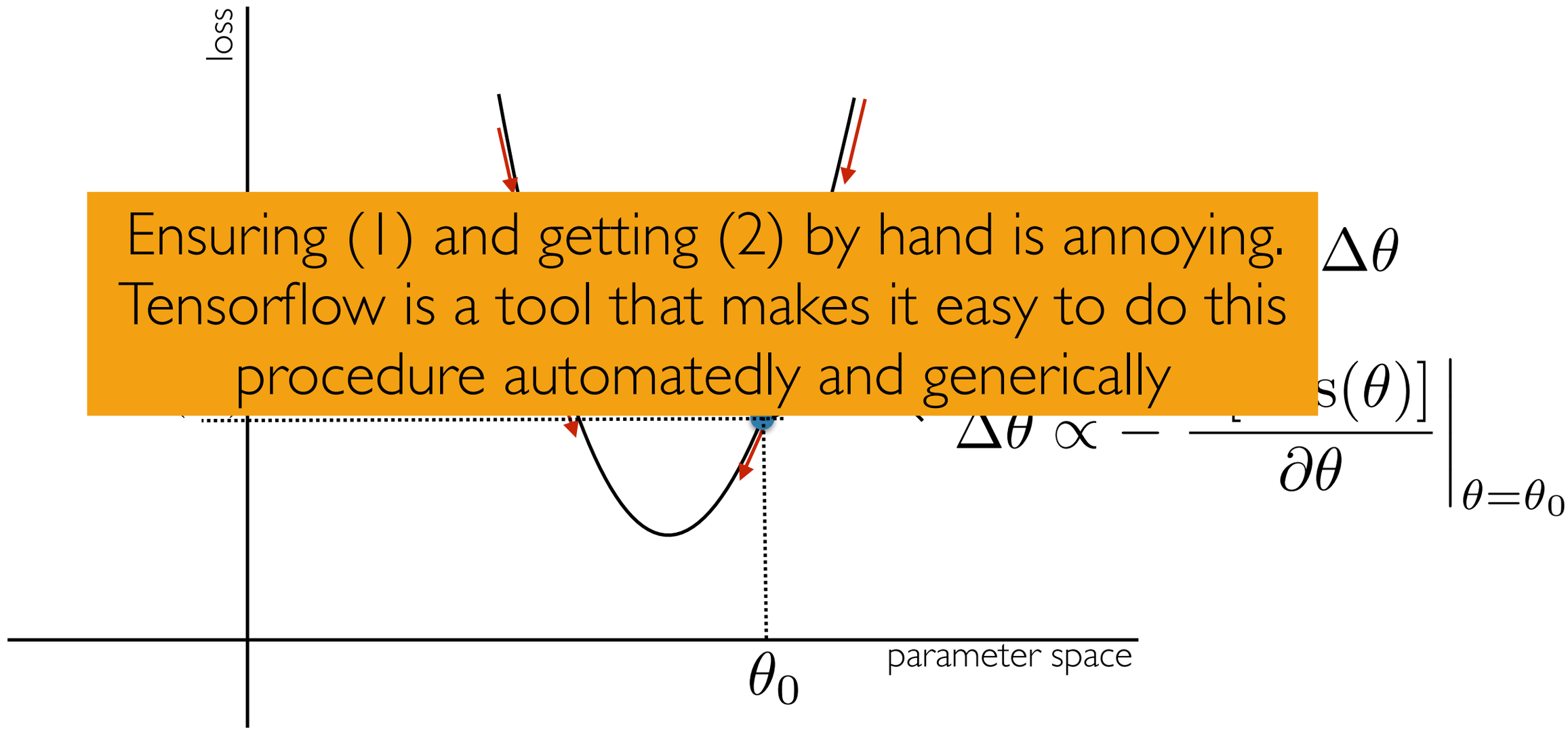
ALSO:

1) loss needs to be differentiable (or close to it) in parameters

2) you actually have to compute the derivative

Optimization

option 3: follow the (negative of) the loss gradient



Ensuring (1) and getting (2) by hand is annoying. Tensorflow is a tool that makes it easy to do this procedure automatedly and generically

ALSO:

1) loss needs to be differentiable (or close to it) in parameters

2) you actually have to compute the derivative

Optimization

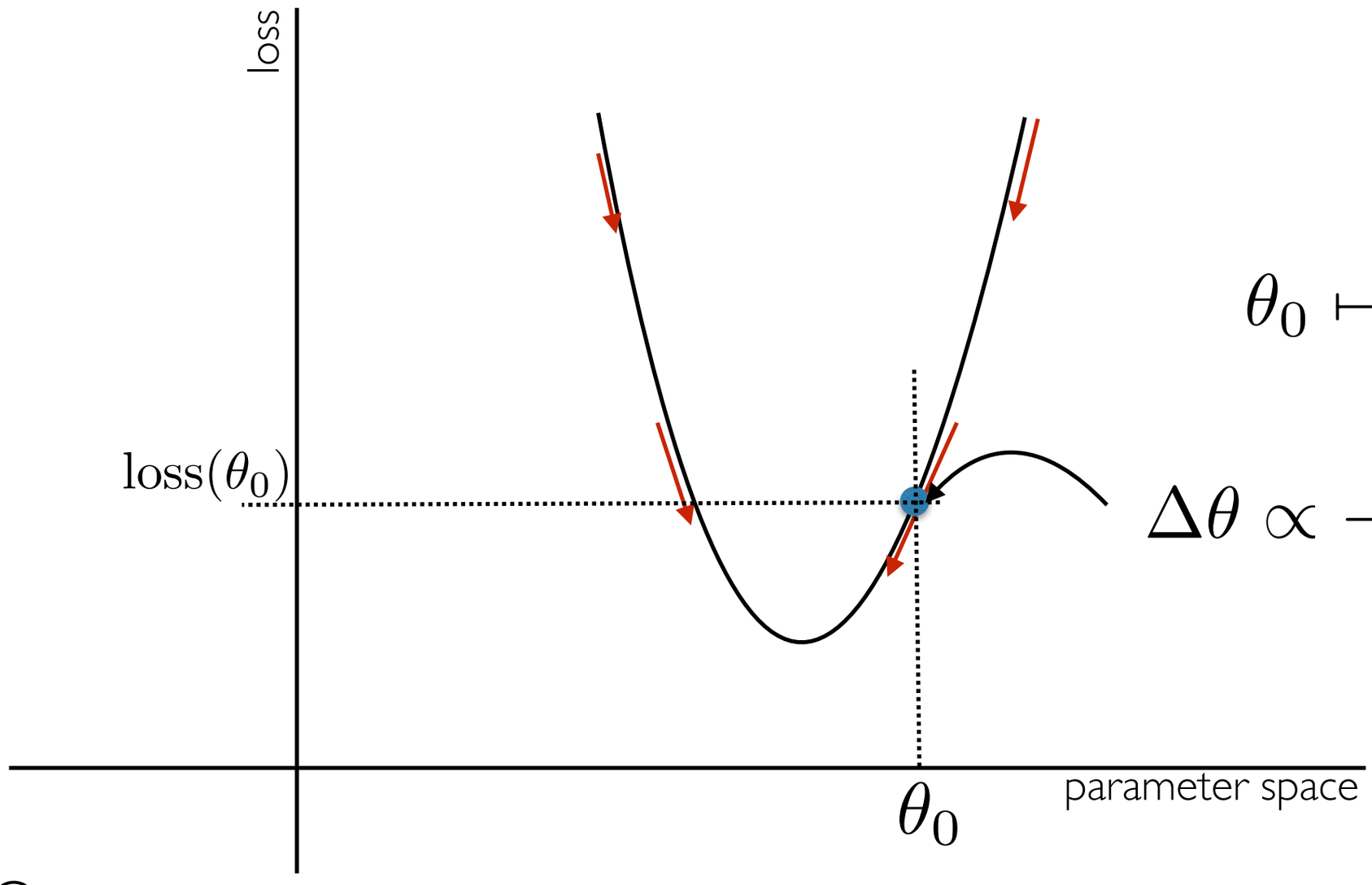
1. Go to Calculus Review slides … and follow along with calculus review code notebook

2. then when that's done, go to Intro to Tensorflow slides and related code notebook

3. … and when *that's* done, come back here and follow the optimization code notebook

Optimization

option 3: follow the (negative of) the loss gradient



$$\theta_0 \mapsto \theta_0 + \Delta\theta$$

$$\Delta\theta \propto -\left.\frac{\partial[\text{loss}(\theta)]}{\partial\theta}\right|_{\theta=\theta_0}$$

ALSO:

1) loss needs to be differentiable (or close to it) in parameters

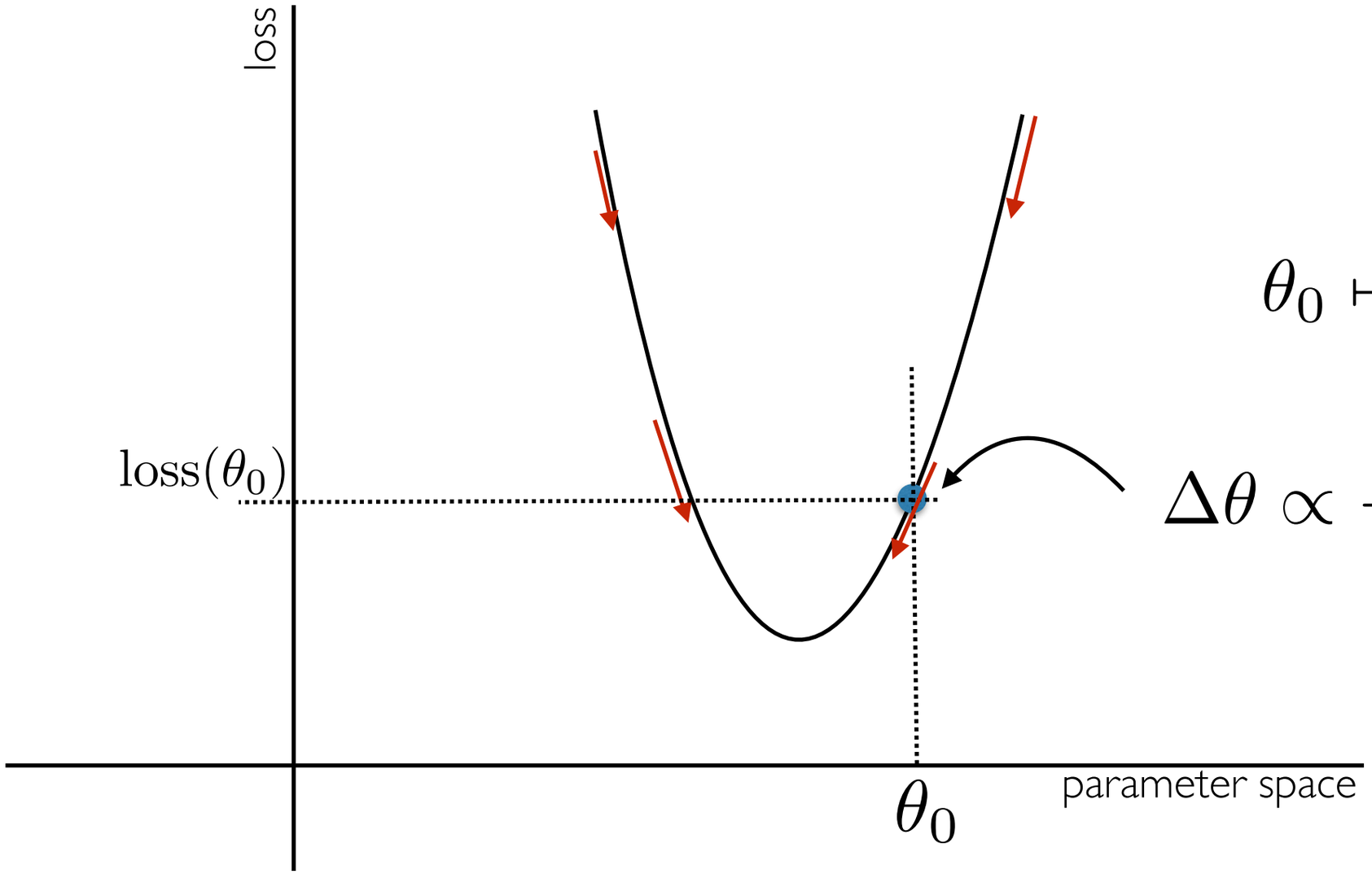
2) you actually have to compute the derivative

Optimization

[ipynb: Gradient Descent]

Gradient Descent

Main problem with gradient-based optimization: instability

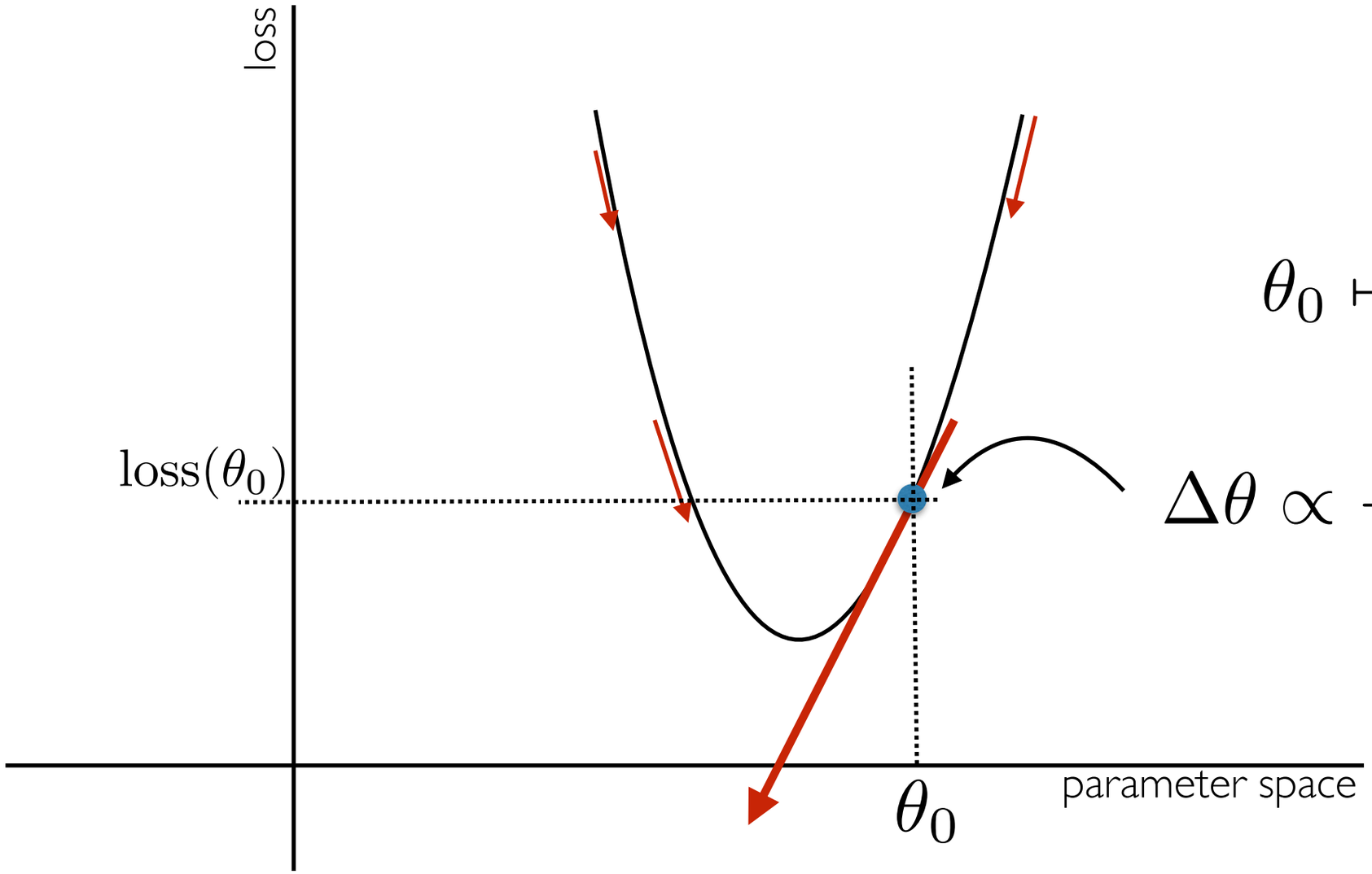


$$\theta_0 \mapsto \theta_0 + \Delta\theta$$

$$\Delta\theta \propto -\left.\frac{\partial[\text{loss}(\theta)]}{\partial\theta}\right|_{\theta=\theta_0}$$

Gradient Descent

Main problem with gradient-based optimization: instability

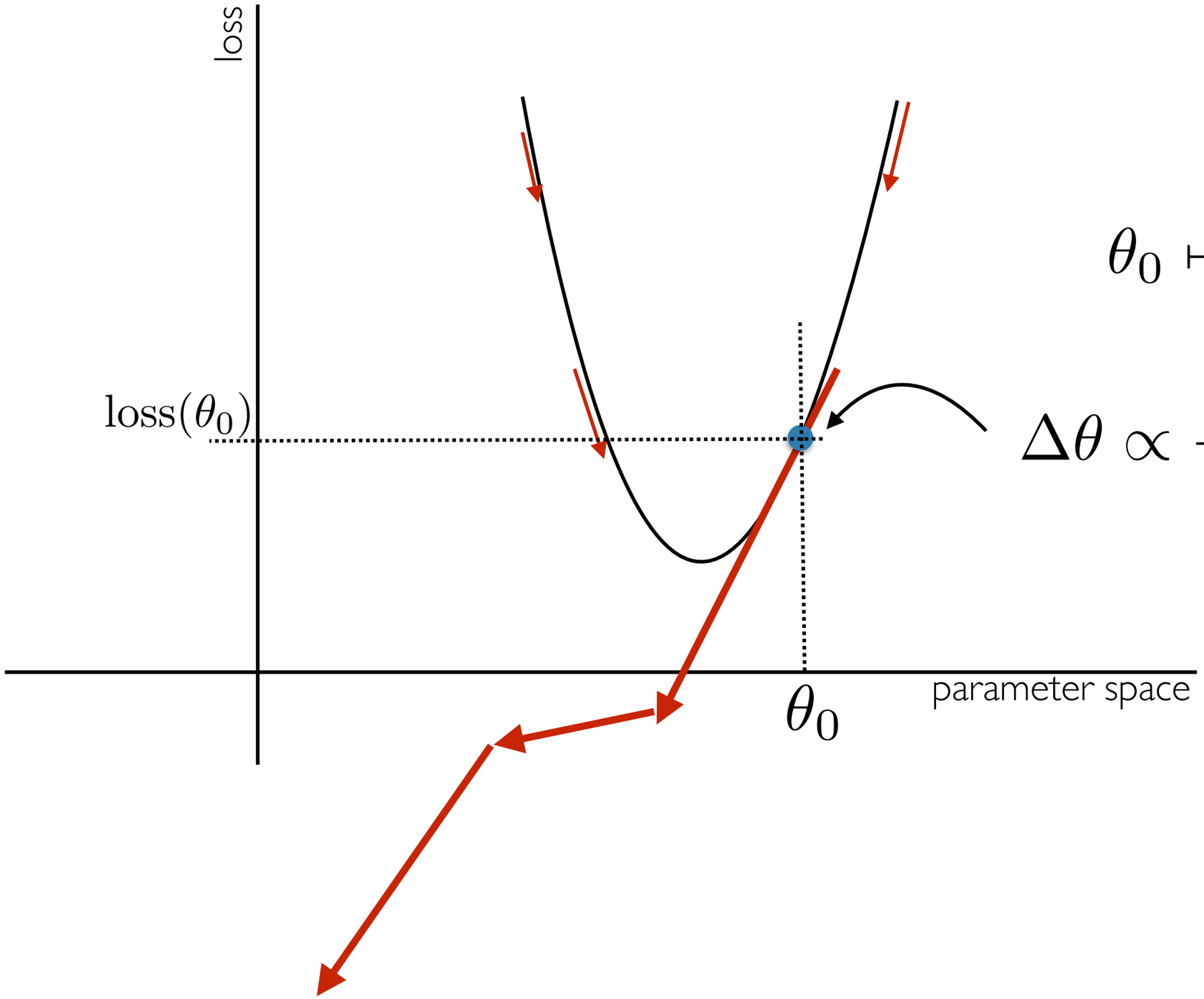


$$\theta_0 \mapsto \theta_0 + \Delta\theta$$

$$\Delta\theta \propto -\left.\frac{\partial[\text{loss}(\theta)]}{\partial\theta}\right|_{\theta=\theta_0}$$

Gradient Descent

Main problem with gradient-based optimization: instability

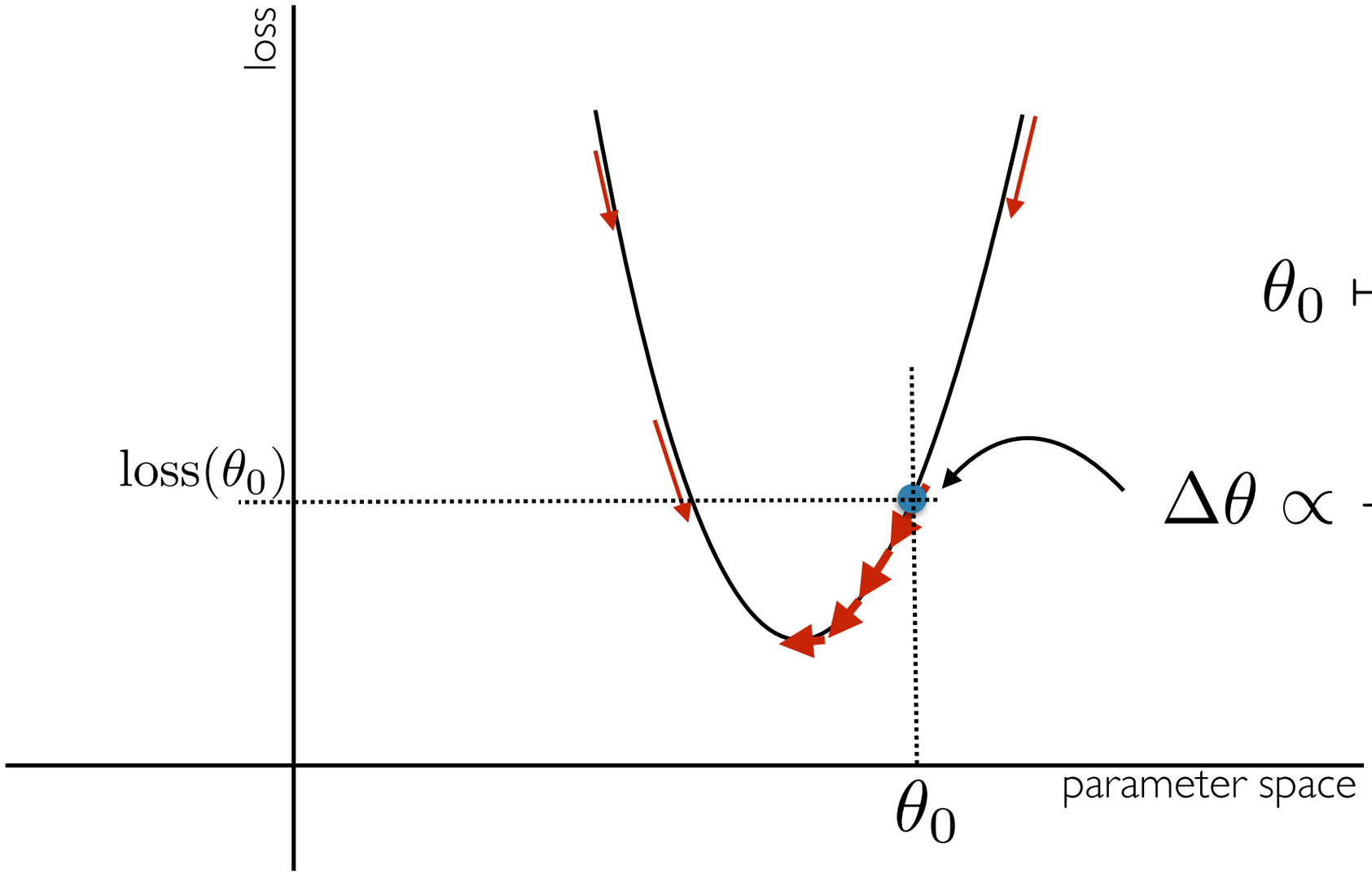


$$\theta_0 \mapsto \theta_0 + \Delta\theta$$

$$\Delta\theta \propto -\left.\frac{\partial[\text{loss}(\theta)]}{\partial\theta}\right|_{\theta=\theta_0}$$

Gradient Descent

Main problem with gradient-based optimization: instability



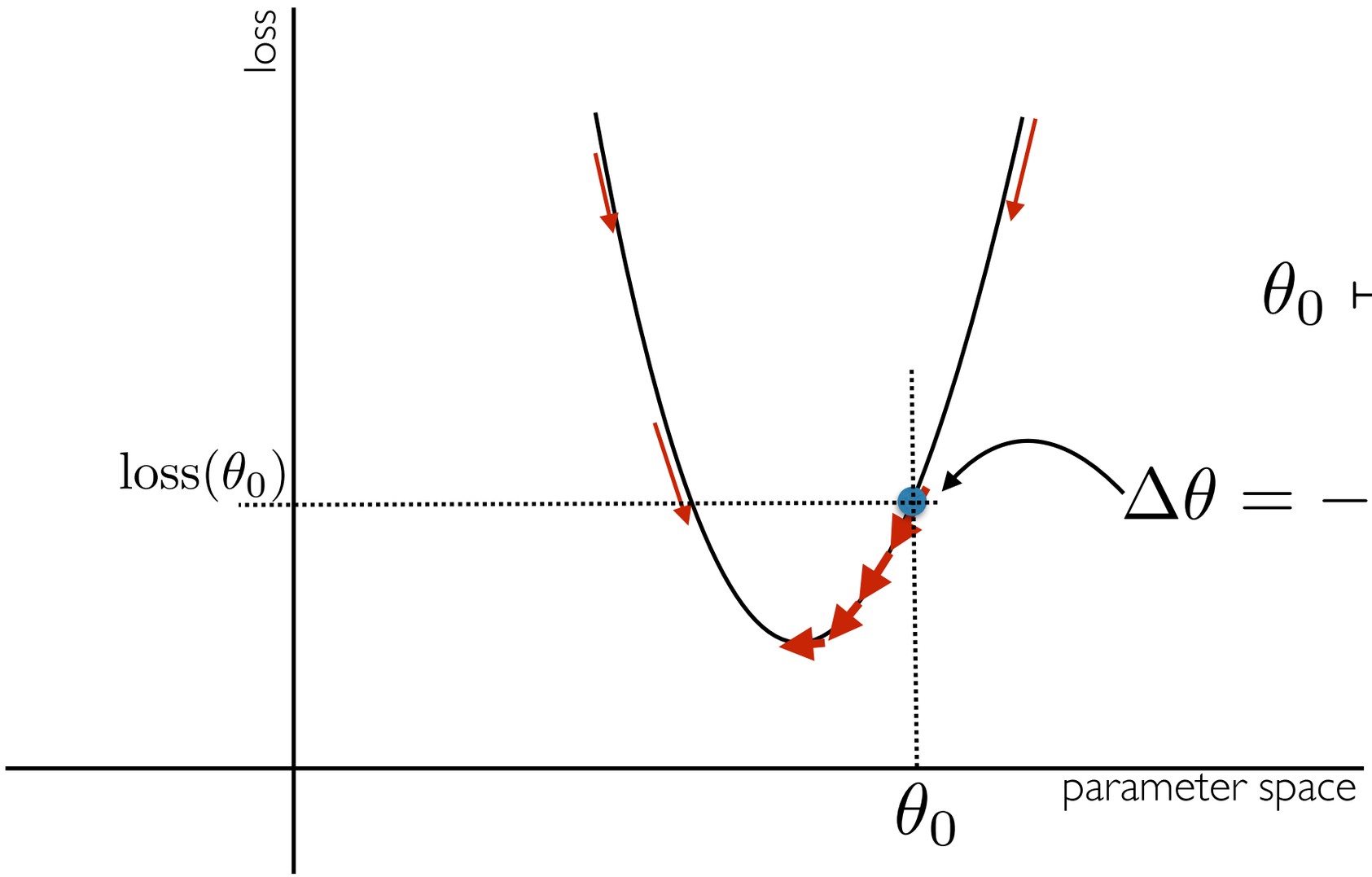
$$\theta_0 \mapsto \theta_0 + \Delta\theta$$

$$\Delta\theta \propto -\left.\frac{\partial[\text{loss}(\theta)]}{\partial\theta}\right|_{\theta=\theta_0}$$

simple solution: take smaller steps

Gradient Descent

Main problem with gradient-based optimization: instability



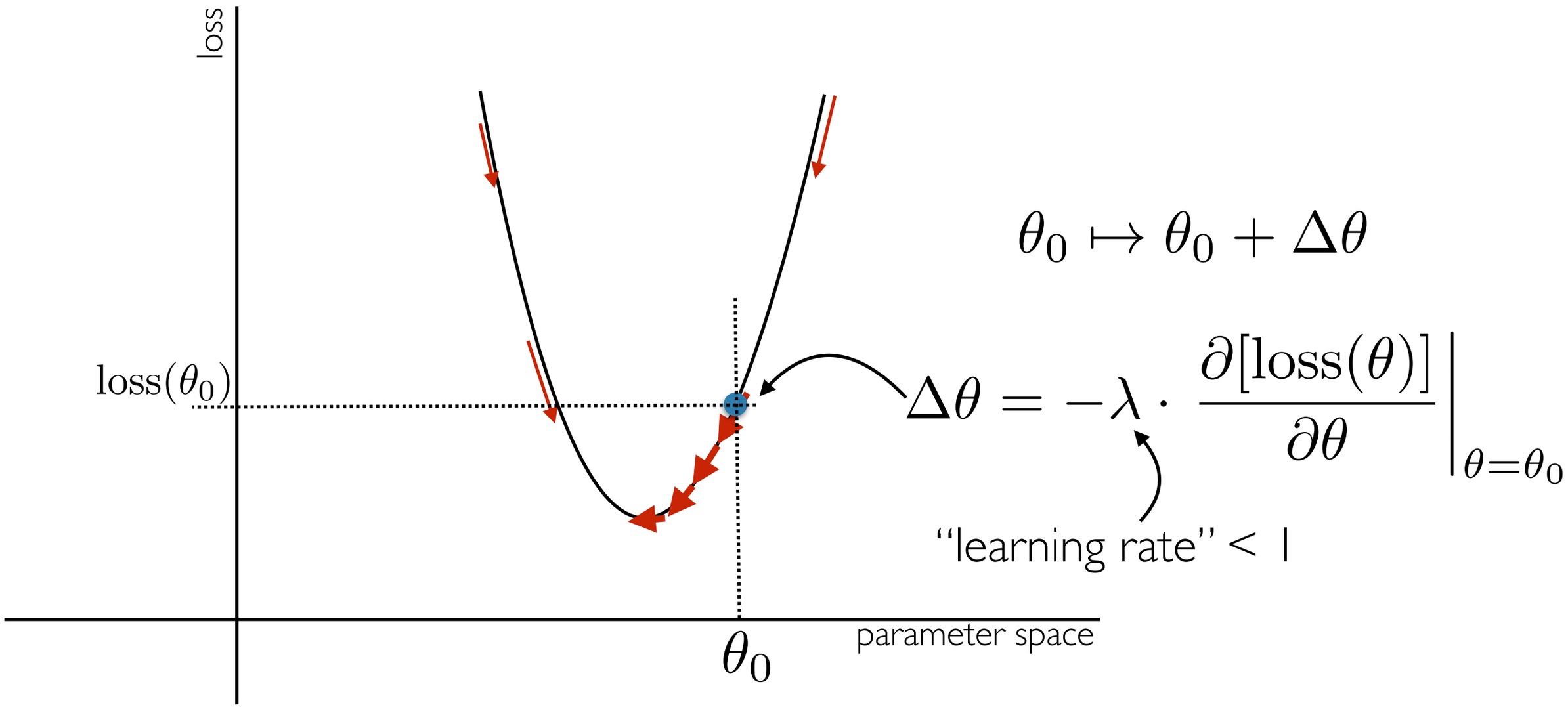
$$\theta_0 \mapsto \theta_0 + \Delta\theta$$

$$\Delta\theta = -\lambda \cdot \left.\frac{\partial[\text{loss}(\theta)]}{\partial\theta}\right|_{\theta=\theta_0}$$

simple solution: take smaller steps

Gradient Descent

Main problem with gradient-based optimization: instability

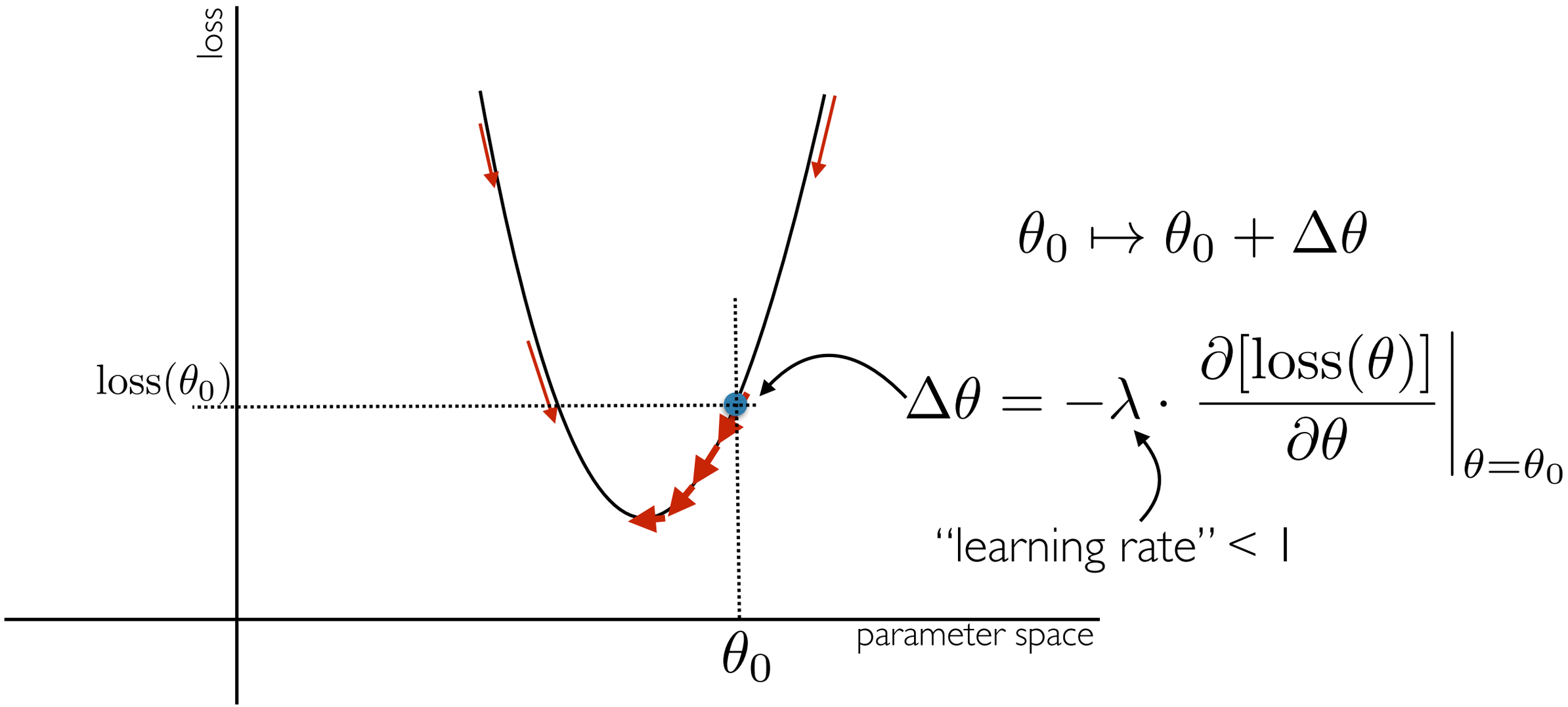


simple solution: take smaller steps

Pro: more stable

Gradient Descent

Main problem with gradient-based optimization: instability



simple solution: take smaller steps

Pro: more stable

Con: if you pick LR too small, converges slowly

Gradient Descent

[Ipynb: With Learning Rate]

Gradient Descent

There are multiple improvements on Gradient Descent. One is called the "Momentum" method

Gradient Descent

There are multiple improvements on Gradient Descent. One is called the "Momentum" method

Original gradient descent:

$$\Delta\theta = -\lambda \cdot \nabla_\theta [\text{loss}(\theta)]|_{\theta=\theta_0}$$

Gradient Descent

There are multiple improvements on Gradient Descent. One is called the "Momentum" method

Original gradient descent:

$$\Delta\theta = -\lambda \cdot \nabla_\theta [\text{loss}(\theta)]|_{\theta=\theta_0}$$

Momentum method:

$$\Delta\theta = -\lambda \cdot \left[\nabla_\theta [\text{loss}(\theta)]|_{\theta=\theta_0} + \mu \cdot \text{grad\_accum} \right]$$

Gradient Descent

There are multiple improvements on Gradient Descent. One is called the "Momentum" method

Original gradient descent:

$$\Delta\theta = -\lambda \cdot \nabla_\theta [\text{loss}(\theta)]|_{\theta=\theta_0}$$

Momentum method:

momentum parameter

$$\Delta\theta = -\lambda \cdot \left[\nabla_\theta [\text{loss}(\theta)]|_{\theta=\theta_0} + \mu \cdot \text{grad\_accum}\right]$$

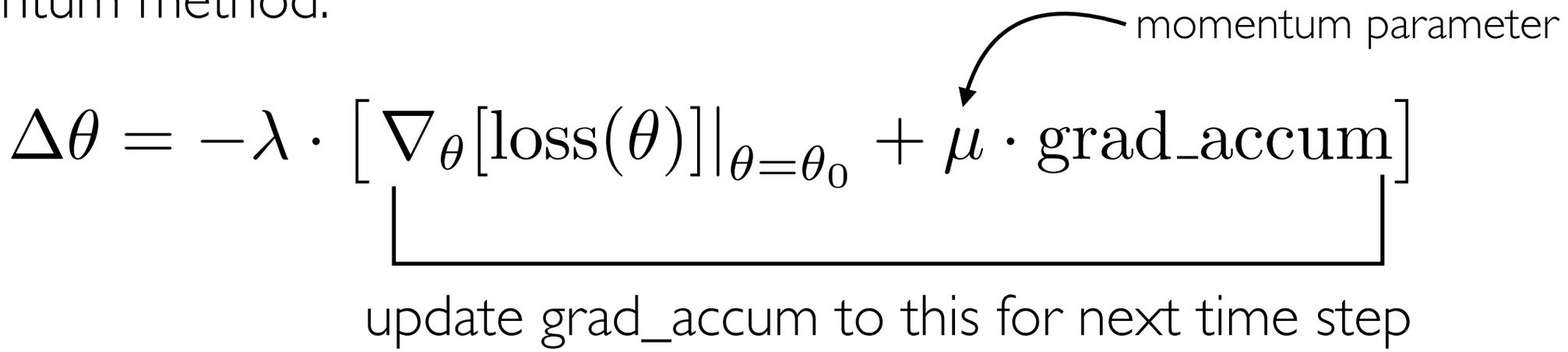
Gradient Descent

There are multiple improvements on Gradient Descent. One is called the "Momentum" method

Original gradient descent:

$$\Delta\theta = -\lambda \cdot \nabla_\theta [\text{loss}(\theta)]|_{\theta=\theta_0}$$

Momentum method:



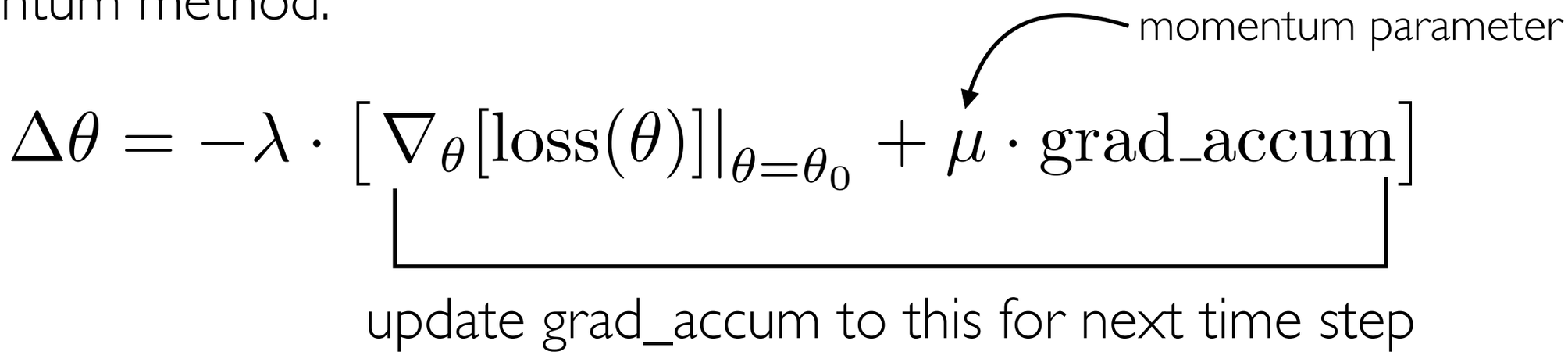
Gradient Descent

There are multiple improvements on Gradient Descent. One is called the "Momentum" method

Original gradient descent:

$$\Delta\theta = -\lambda \cdot \nabla_\theta [\text{loss}(\theta)]|_{\theta=\theta_0}$$

Momentum method:



Pros: faster, stabler convergence.

Cons: nothing (relative to not doing it).

Gradient Descent

Amazing explanation of why momentum is so good:

https://distill.pub/2017/momentum/

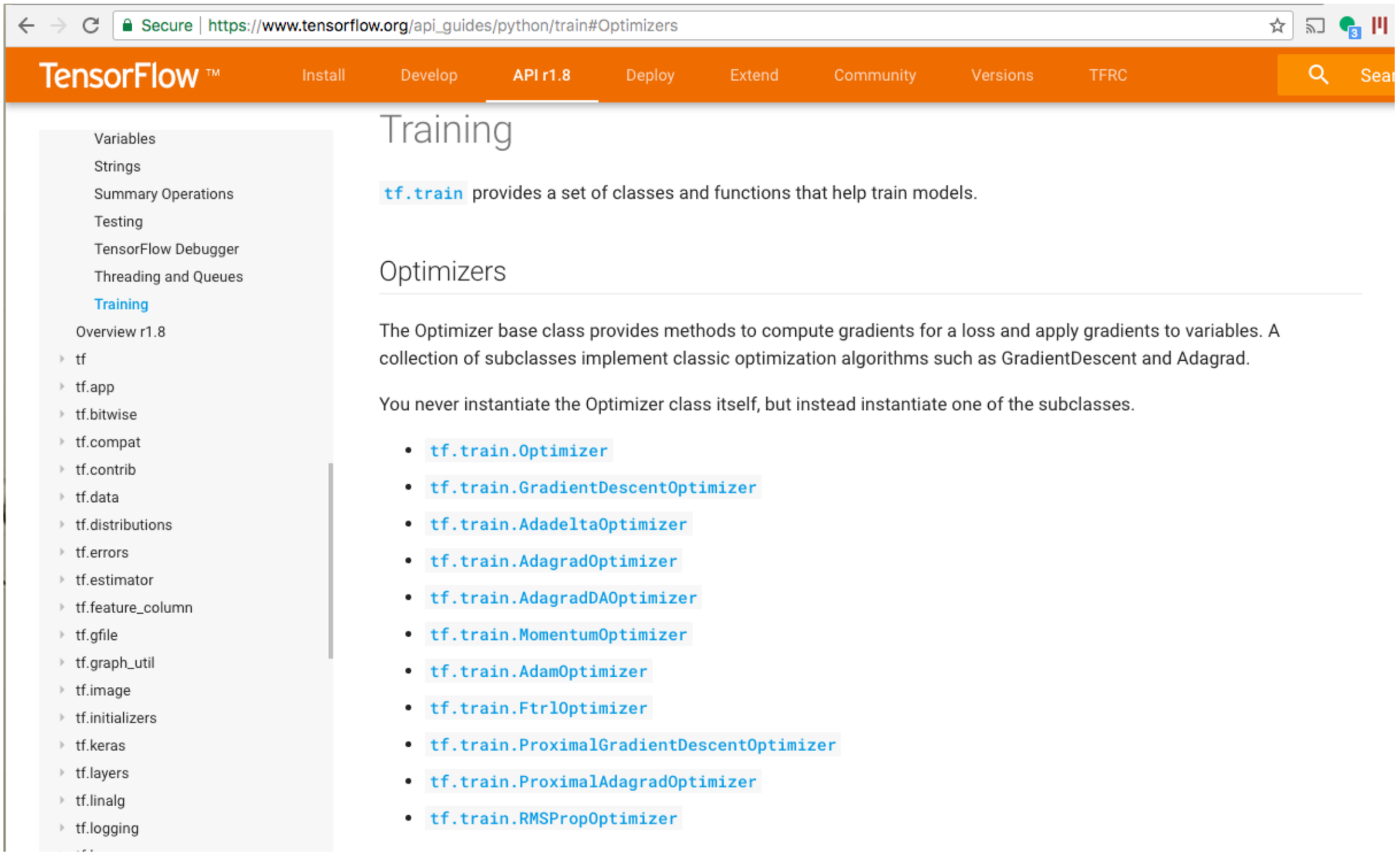
Ours:	Theirs:
λ	α
μ	β

"Here's a popular story about momentum: gradient descent is a man walking down a hill. He follows the steepest path downwards; his progress is slow, but steady. Momentum is a heavy ball rolling down the same hill. The added inertia acts both as a smoother and an accelerator, dampening oscillations and causing us to barrel through narrow valleys, small humps and local minima. …."

Gradient Descent

[Ipynb: Tensorflow's Built-in Optimizers]

Gradient Descent



Second-Order Methods

Taylor series for function ***f***:

Second-Order Methods

Taylor series for function ***f***: $x = x_0 + \Delta x$

$$f(x) = f(x_0) + f'(x_0)\Delta x + \frac{1}{2} f''(x_0)\Delta x^2 + \dots$$

Second-Order Methods

Taylor series for function ***f***: $x = x_0 + \Delta x$

$$f(x_0, \Delta x) = f(x_0) + f'(x_0)\Delta x + \frac{1}{2} f''(x_0)\Delta x^2 + \dots$$

Second-Order Methods

Taylor series for function ***f***: $x = x_0 + \Delta x$

$$f(x_0, \Delta x) = f(x_0) + f'(x_0)\Delta x + \frac{1}{2} f''(x_0)\Delta x^2 + \dots$$

Differentiate by ***Δx***:

Second-Order Methods

Taylor series for function ***f***: $x = x_0 + \Delta x$

$$f(x_0, \Delta x) = f(x_0) + f'(x_0)\Delta x + \frac{1}{2} f''(x_0)\Delta x^2 + \dots$$

Differentiate by ***Δx***:

$$\frac{\partial f(x_0, \Delta x)}{\partial \Delta x} = f'(x_0) + f''(x_0)\Delta x + \dots$$

Second-Order Methods

Taylor series for function ***f***: $x = x_0 + \Delta x$

$$f(x_0, \Delta x) = f(x_0) + f'(x_0)\Delta x + \frac{1}{2} f''(x_0)\Delta x^2 + \dots$$

Differentiate by ***Δx***:

$$\frac{\partial f(x_0, \Delta x)}{\partial \Delta x} = f'(x_0) + f''(x_0)\Delta x + \dots$$

Set to 0 and ignore HOT:

Second-Order Methods

Taylor series for function ***f***: $x = x_0 + \Delta x$

$$f(x_0, \Delta x) = f(x_0) + f'(x_0)\Delta x + \frac{1}{2} f''(x_0)\Delta x^2 + \dots$$

Differentiate by ***Δx***:

$$\frac{\partial f(x_0, \Delta x)}{\partial \Delta x} = f'(x_0) + f''(x_0)\Delta x + \dots$$

Set to 0 and ignore HOT:

$$0 = f'(x_0) + f''(x_0)\Delta x$$

Second-Order Methods

Taylor series for function ***f***: $x = x_0 + \Delta x$

$$f(x_0, \Delta x) = f(x_0) + f'(x_0)\Delta x + \frac{1}{2} f''(x_0)\Delta x^2 + \dots$$

Differentiate by ***Δx***:

$$\frac{\partial f(x_0, \Delta x)}{\partial \Delta x} = f'(x_0) + f''(x_0)\Delta x + \dots$$

Set to 0 and ignore HOT:

$$0 = f'(x_0) + f''(x_0)\Delta x$$

Solve for ***Δx***:

Second-Order Methods

Taylor series for function ***f***: $x = x_0 + \Delta x$

$$f(x_0, \Delta x) = f(x_0) + f'(x_0)\Delta x + \frac{1}{2} f''(x_0)\Delta x^2 + \dots$$

Differentiate by ***Δx***:

$$\frac{\partial f(x_0, \Delta x)}{\partial \Delta x} = f'(x_0) + f''(x_0)\Delta x + \dots$$

Set to 0 and ignore Higher Order Terms (HOT):

$$0 = f'(x_0) + f''(x_0)\Delta x$$

Solve for ***Δx***:

$$\Delta x = -\frac{f'(x_0)}{f''(x_0)}$$

Second-Order Methods

Taylor series for function ***f***: $x = x_0 + \Delta x$

$$f(x_0, \Delta x) = f(x_0) + f'(x_0)\Delta x + \frac{1}{2} f''(x_0)\Delta x^2 + \dots$$

Differentiate by ***Δx***:

$$\frac{\partial f(x_0, \Delta x)}{\partial \Delta x} = f'(x_0) + f''(x_0)\Delta x + \dots$$

Set to 0 and ignore HOT:

$$0 = f'(x_0) + f''(x_0)\Delta x$$

Solve for ***Δx***:

$$\Delta x = -f'(x_0) \cdot \boxed{\frac{1}{f''(x_0)}}$$

really correct value for learning rate λ

Second-Order Methods

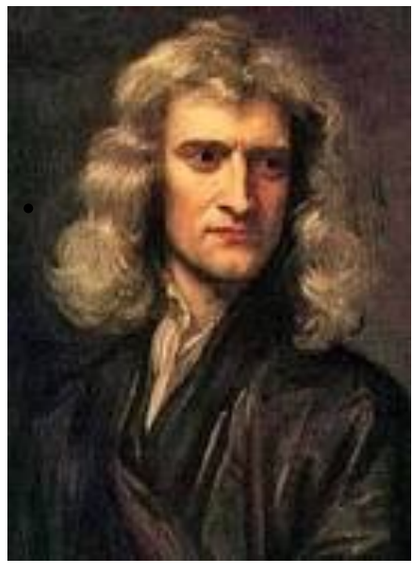
Taylor series for function ***f***: $x = x_0 + \Delta x$

$$f(x_0, \Delta x) = f(x_0) + f'(x_0)\Delta x + \frac{1}{2} f''(x_0)\Delta x^2 + \dots$$

Differentiate by ***Δx***:

$$\frac{\partial f(x_0, \Delta x)}{\partial \Delta x} = f'(x_0) + f''(x_0)\Delta x + \dots$$

Newton-Raphson



Set to 0 and ignore HOT:

$$0 = f'(x_0) + f''(x_0)\Delta x$$

Solve for ***Δx***:

$$\Delta x = -f'(x_0) \cdot \boxed{\frac{1}{f''(x_0)}}$$

really correct value for learning rate λ

Second-Order Methods

Taylor series for function ***f(x1, ... xn)*** of ***n*** inputs:

$$f(\vec{x}_0, \Delta\vec{x}) = f(\vec{x}_0) + \nabla f(\vec{x}_0) * \Delta\vec{x} + \frac{1}{2}(H[F] \cdot \vec{x}_0) * \Delta x^2 + \dots$$

the gradient $$\nabla f = \left(\frac{\partial f}{\partial x_1}, \dots, \frac{\partial f}{\partial x_n} \right)$$

the "hessian" matrix

$$H[f] = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1 \partial x_1} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \frac{\partial^2 f}{\partial x_1 \partial x_3} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2 \partial x_2} & \frac{\partial^2 f}{\partial x_2 \partial x_3} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \frac{\partial^2 f}{\partial x_n \partial x_3} & \cdots & \frac{\partial^2 f}{\partial x_n \partial x_n} \end{bmatrix}$$

shape = (***n***, ***n***)

Second-Order Methods

Taylor series for function ***f(x1, ... xn)*** of ***n*** inputs:

$$f(\vec{x}_0, \Delta\vec{x}) = f(\vec{x}_0) + \nabla f(\vec{x}_0) * \Delta\vec{x} + \frac{1}{2}(H[F] \cdot \vec{x}_0) * \Delta x^2 + \dots$$

matrix-on-vector multiplication

the gradient $\nabla f = \left(\frac{\partial f}{\partial x_1}, \dots, \frac{\partial f}{\partial x_n} \right)$

the "hessian" matrix

$$H[f] = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1 \partial x_1} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \frac{\partial^2 f}{\partial x_1 \partial x_3} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2 \partial x_2} & \frac{\partial^2 f}{\partial x_2 \partial x_3} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \frac{\partial^2 f}{\partial x_n \partial x_3} & \cdots & \frac{\partial^2 f}{\partial x_n \partial x_n} \end{bmatrix}$$

shape = (n, n)

Second-Order Methods

Taylor series for function ***f(x1, ... xn)*** of ***n*** inputs:

$$f(\vec{x}_0, \Delta\vec{x}) = f(\vec{x}_0) + \nabla f(\vec{x}_0) * \Delta\vec{x} + \frac{1}{2}(H[F] \cdot \vec{x}_0) * \Delta x^2 + \dots$$

single variable:

$$\Delta x = -\frac{f'(x_0)}{f''(x_0)}$$

$$\Rightarrow$$

multi-variable:

$$\Delta\vec{x} = -(H[f])^{-1} \cdot \nabla f(\vec{x}_0)$$

Second-Order Methods

Taylor series for function ***f(x1, ... xn)*** of ***n*** inputs:

$$f(\vec{x}_0, \Delta\vec{x}) = f(\vec{x}_0) + \nabla f(\vec{x}_0) * \Delta\vec{x} + \frac{1}{2}(H[F] \cdot \vec{x}_0) * \Delta x^2 + \dots$$

single variable:

$$\Delta x = -\frac{f'(x_0)}{f''(x_0)}$$

$\Rightarrow$

multi-variable:

$$\Delta\vec{x} = -(H[f])^{-1} \cdot \nabla f(\vec{x}_0)$$

n vector

nxn matrix

n vector

matrix inverse

matrix-on-vector mult

Gradient Descent

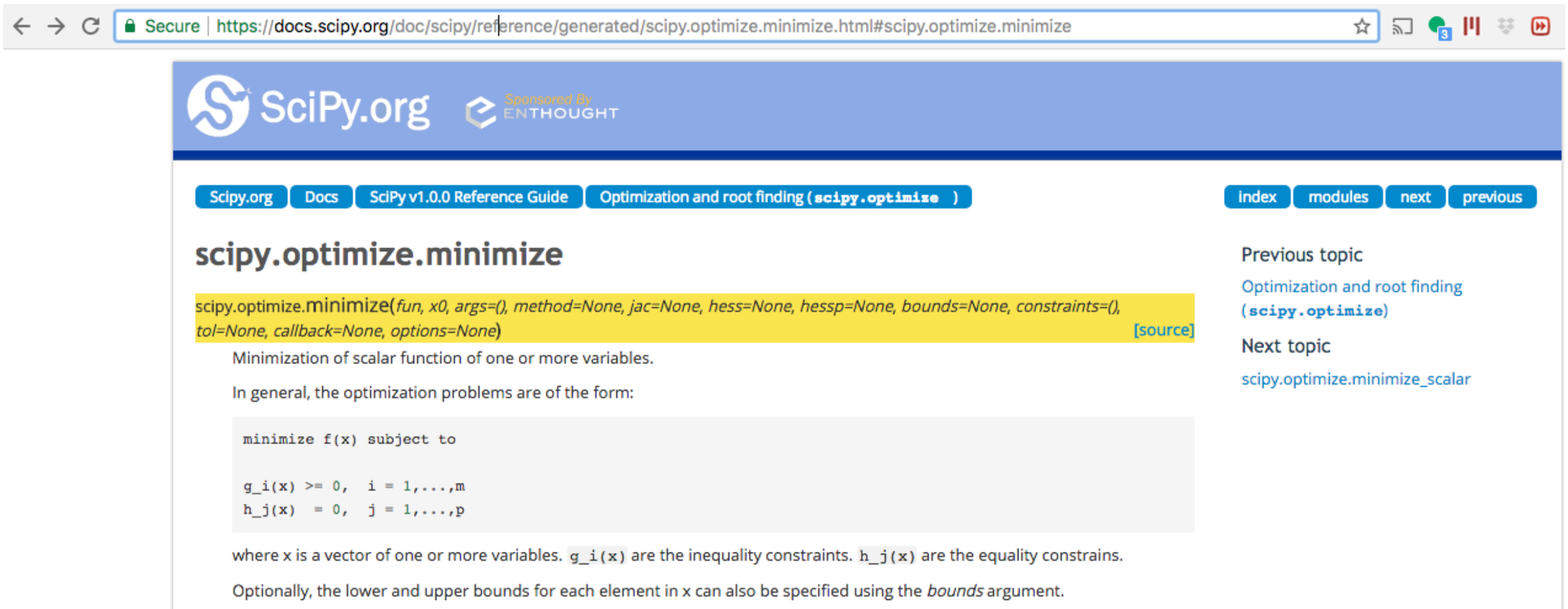
[Ipynb: Newton's Method]

External Optimizers

Don't have to use optimizers just from Tensorflow,

… even if using Tensorflow to get derivatives

Really powerful general optimizer for general Python use:



External Optimizers

```
def minimize(loss_func, x0, optimizer_type)
```

starts at x0, uses whatever specified optimizer to minimize **`loss_func`**

doesn't use derivative info

External Optimizers

```
def minimize(loss_func, x0, optimizer_type)
```

starts at x0, uses whatever specified optimizer to minimize **`loss_func`**

doesn't use derivative info

```
def minimize(loss_func, x0, optimizer_type, jac=True)
```

loss_func must now return both loss and loss gradient:

```
loss_val, grad = loss_func(x)
```

does use derivative info

External Optimizers

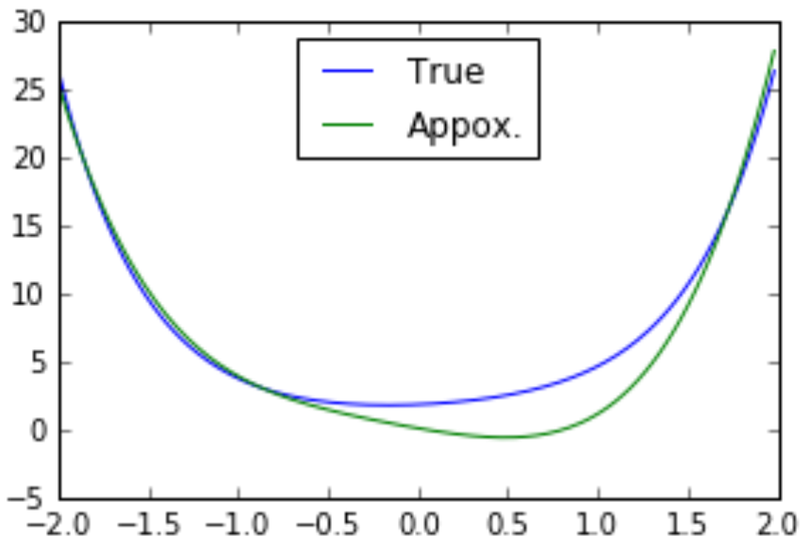
[Ipynb: Using External Optimizers]

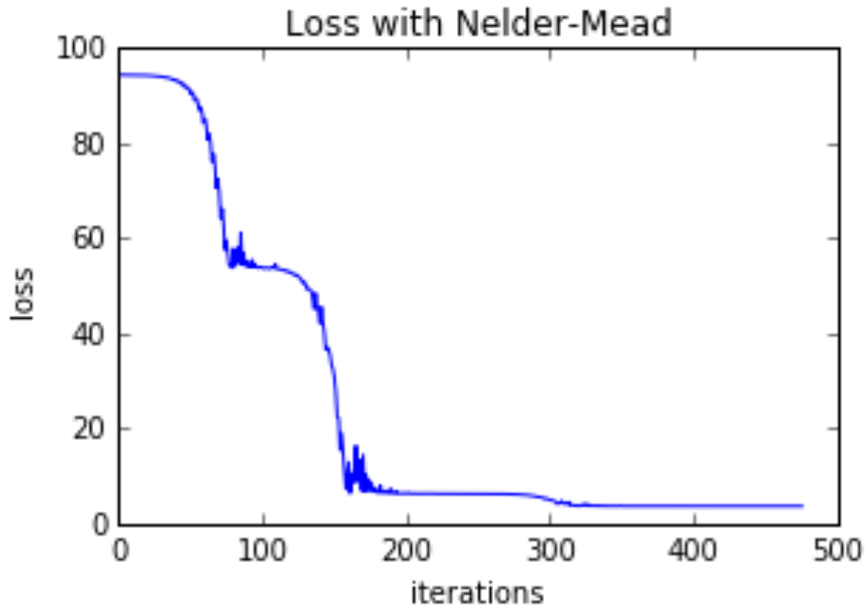
External Optimizers

Target for learning (with 5-th order polynomial model)

*y = np.cosh(2 * x) + np.sin(x + 1)*

Not using derivative info:





Using derivative: