人工智能实践
Artificial Intelligence Practice

DCS3015 Autumn 2022

Chao Yu（余超）

School of Computer Science and Engineering
Sun Yat-Sen University

Lecture 4：机器学习-1

What is Machine Learning?

"Learning is any process by which a system improves performance from experience."

- Herbert Simon

Definition by Tom Mitchell (1998):

Machine Learning is the study of algorithms that

- improve their performance P
- at some task T
- with experience E.

A well-defined learning task is given by <P, T, E>.

Defining the Learning Task

Improve on task T, with respect to
performance metric P, based on experience E

T: Playing checkers
P: Percentage of games won against an arbitrary opponent
E: Playing practice games against itself

T: Recognizing hand-written words
P: Percentage of words correctly classified
E: Database of human-labeled images of handwritten words

T: Driving on four-lane highways using vision sensors
P: Average distance traveled before a human-judged error
E: A sequence of images and steering commands recorded while observing a human driver.

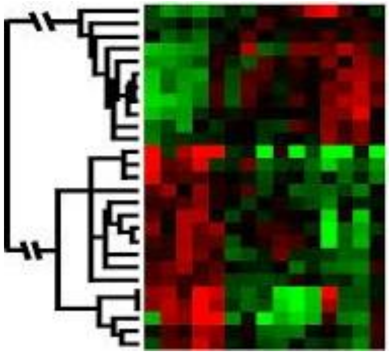
T: Categorize email messages as spam or legitimate.
P: Percentage of email messages correctly classified.
E: Database of emails, some with human-given labels

When Do We Use Machine Learning?

ML is used when:

- Human expertise does not exist (navigating on Mars)
- Humans can’t explain their expertise (speech recognition)
- Models must be customized (personalized medicine)
- Models are based on huge amounts of data (genomics)



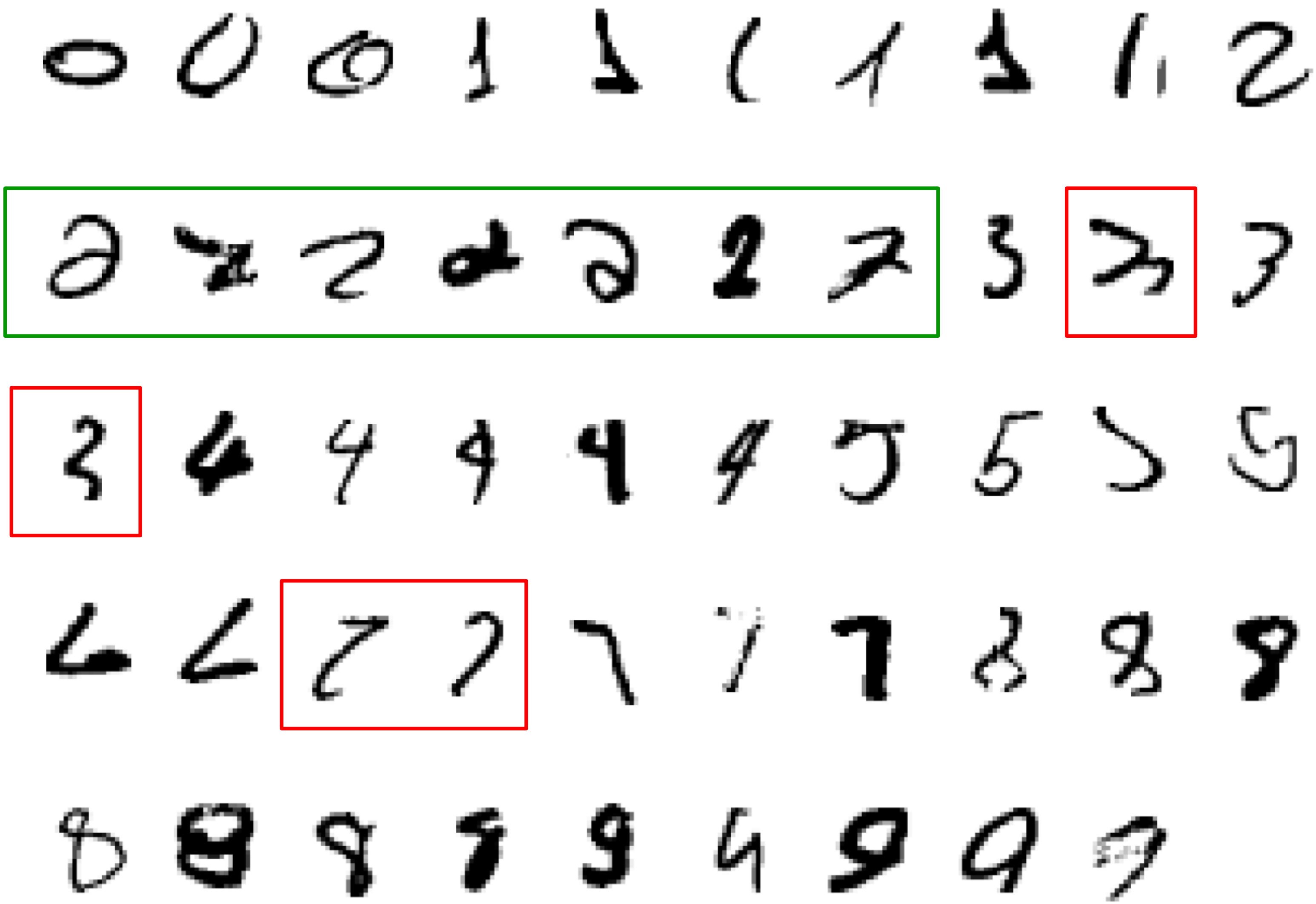



Learning isn’t always useful:

- There is no need to “learn” to calculate payroll

A classic example that requires machine learning:

It is very hard to say what makes a 2



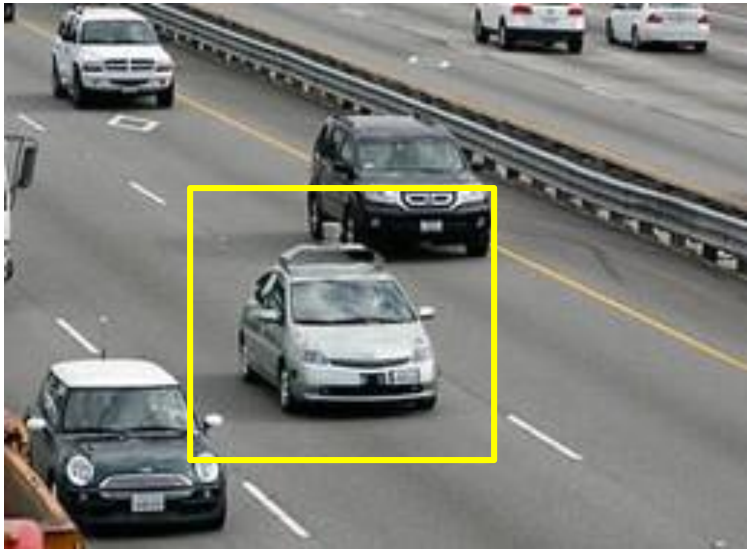
Some more examples that are best solved by using a learning algorithm

- Recognizing patterns:
  - Facial identities or facial expressions
  - Handwritten or spoken words
  - Medical images
- Generating patterns:
  - Generating images or motion sequences
- Recognizing anomalies:
  - Unusual credit card transactions
  - Unusual patterns of sensor readings in a nuclear power plant
- Prediction:
  - Future stock prices or currency exchange rates

Sample Applications

- Web search
- Computational biology
- Finance
- E-commerce
- Space exploration
- Robotics
- Information extraction
- Social networks
- Debugging software
- [Your favorite area]

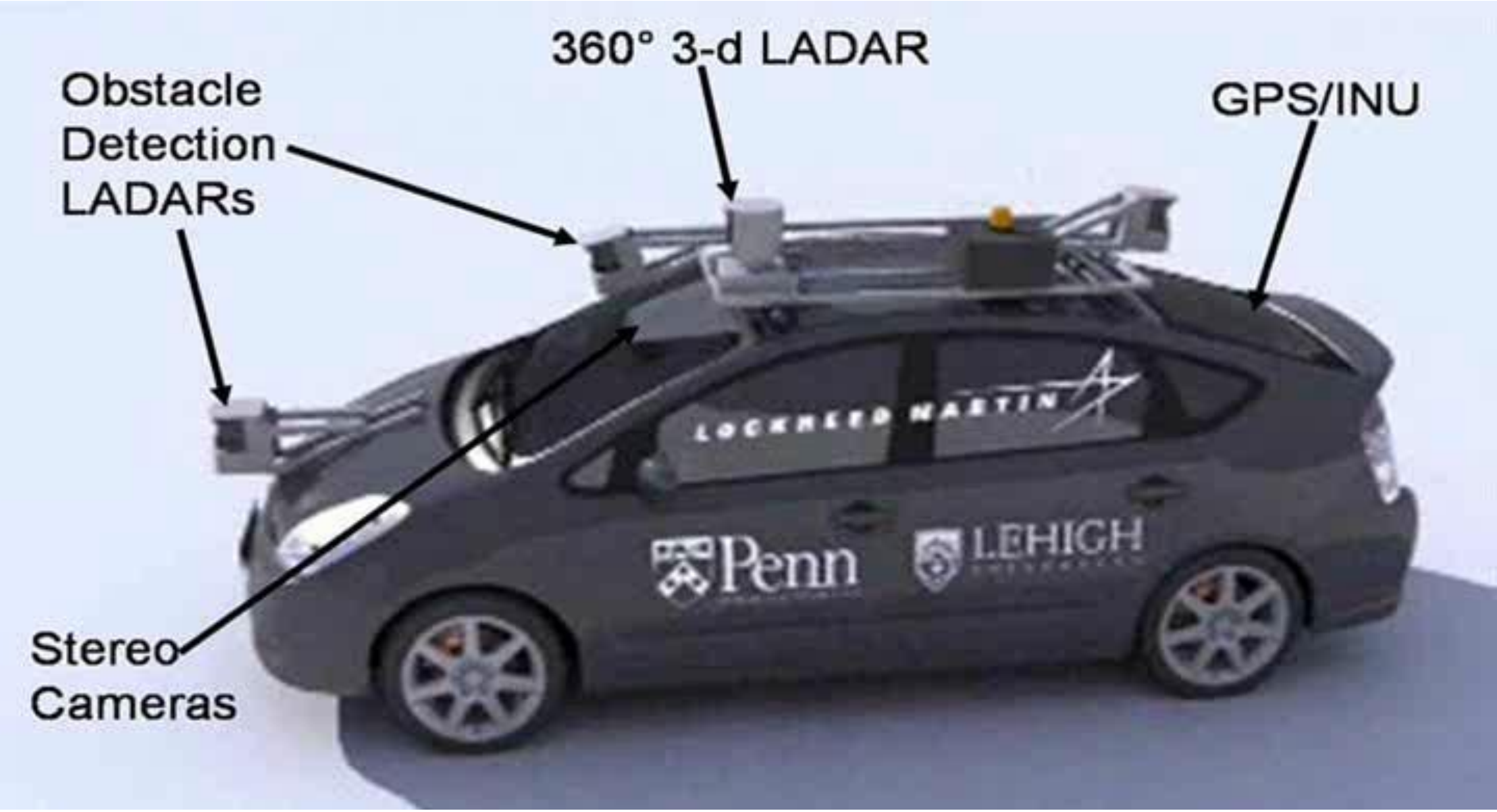
Autonomous Cars



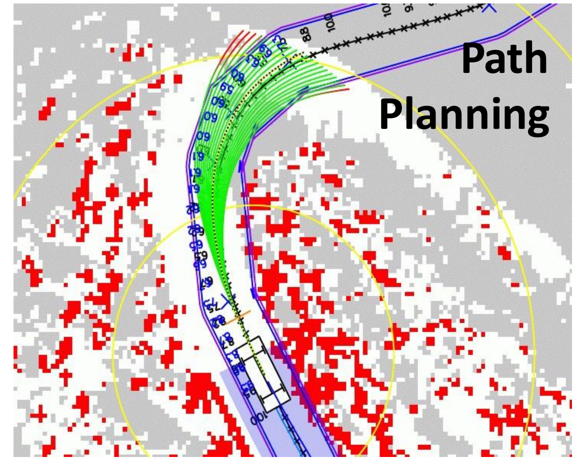
- Nevada made it legal for autonomous cars to drive on roads in June 2011
- As of 2013, four states (Nevada, Florida, California, and Michigan) have legalized autonomous cars

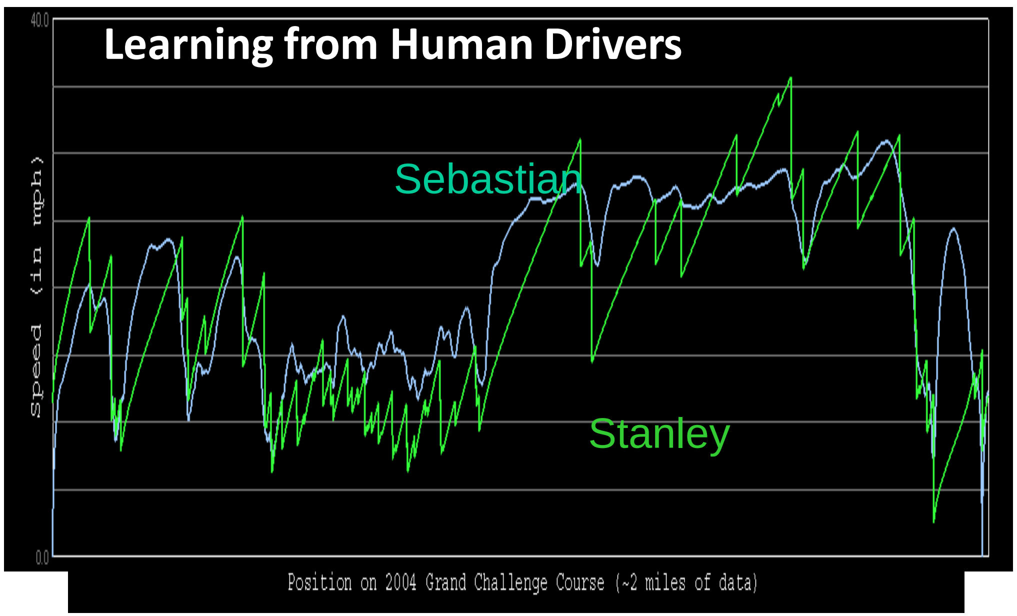
Penn's Autonomous Car →
(Ben Franklin Racing Team)

Autonomous Car Sensors



Autonomous Car Technology





Images and movies taken from Sebastian Thrun's multimedia website.

History of Machine Learning

- 1950s
  - Samuel’s checker player
  - Selfridge’s Pandemonium
- 1960s:
  - Neural networks: Perceptron
  - Pattern recognition
  - Learning in the limit theory
  - Minsky and Papert prove limitations of Perceptron
- 1970s:
  - Symbolic concept induction
  - Winston’s arch learner
  - Expert systems and the knowledge acquisition bottleneck
  - Quinlan’s ID3
  - Michalski’s AQ and soybean diagnosis
  - Scientific discovery with BACON
  - Mathematical discovery with AM

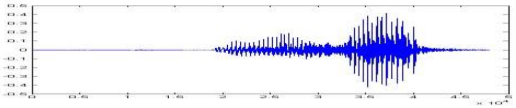
History of Machine Learning (cont.)

- 1980s:
  - Advanced decision tree and rule learning
  - Explanation-based Learning (EBL)
  - Learning and planning and problem solving
  - Utility problem
  - Analogy
  - Cognitive architectures
  - Resurgence of neural networks (connectionism, backpropagation)
  - Valiant's PAC Learning Theory
  - Focus on experimental methodology
- 1990s
  - Data mining
  - Adaptive software agents and web applications
  - Text learning
  - Reinforcement learning (RL)
  - Inductive Logic Programming (ILP)
  - Ensembles: Bagging, Boosting, and Stacking
  - Bayes Net learning

History of Machine Learning (cont.)

- 2000s
  - Support vector machines & kernel methods
  - Graphical models
  - Statistical relational learning
  - Transfer learning
  - Sequence labeling
  - Collective classification and structured outputs
  - Computer Systems Applications (Compilers, Debugging, Graphics, Security)
  - E-mail management
  - Personalized assistants that learn
  - Learning in robotics and vision
- 2010s
  - Deep learning systems
  - Learning for big data
  - Bayesian methods
  - Multi-task & lifelong learning
  - Applications to vision, speech, social networks, learning to read, etc.
  - ???

Machine Learning
≈ Looking for Function

$f($  $) =$ "How are you"

$f($  $) =$ "Cat"

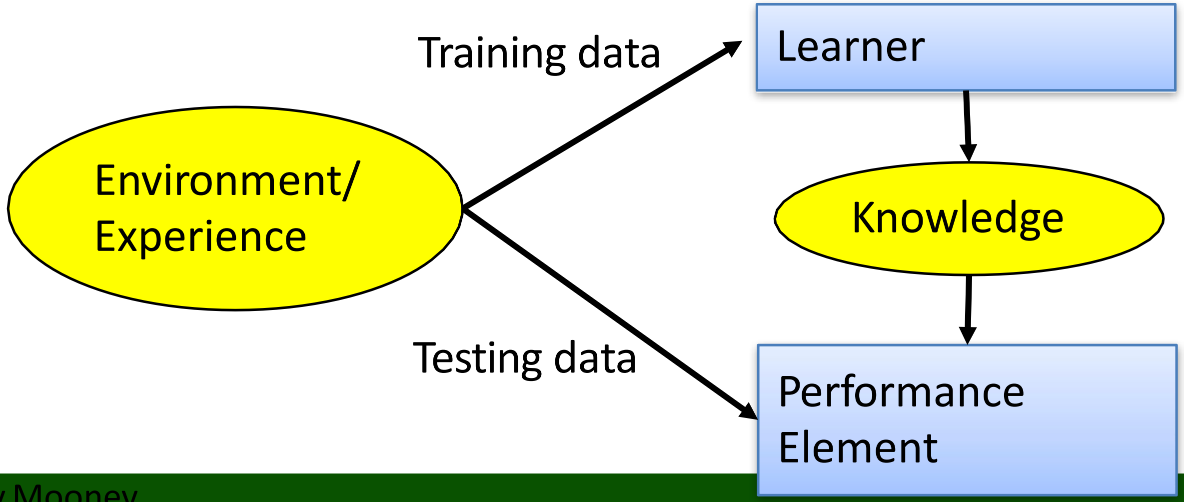
$f($  $) =$ "5-5" (next move)

Types of Learning

- **Supervised (inductive) learning**
  - Given: training data + desired outputs (labels)
- **Unsupervised learning**
  - Given: training data (without desired outputs)
- **Semi-supervised learning**
  - Given: training data + a few desired outputs
- **Reinforcement learning**
  - Rewards from sequence of actions

Designing a Learning System

- Choose the training experience
- Choose exactly what is to be learned
  - i.e. the ***target function***
- Choose how to represent the target function
- Choose a learning algorithm to infer the target function from the experience



Training vs. Test Distribution

- We generally assume that the training and test examples are independently drawn from the same overall distribution of data
  - We call this “i.i.d” which stands for “independent and identically distributed”

- If examples are not independent, requires ***collective classification***

- If test distribution is different, requires ***transfer learning***

ML in a Nutshell

- Tens of thousands of machine learning algorithms
  - Hundreds new every year

- Every ML algorithm has three components:
  - **Representation（Model）**
  - **Optimization**
  - **Evaluation**

Various Function Representations

- Numerical functions
  - Linear regression
  - Neural networks
  - Support vector machines
- Symbolic functions
  - Decision trees
  - Rules in propositional logic
  - Rules in first-order predicate logic
- Instance-based functions
  - Nearest-neighbor
  - Case-based
- Probabilistic Graphical Models
  - Naïve Bayes
  - Bayesian networks
  - Hidden-Markov Models (HMMs)
  - Probabilistic Context Free Grammars (PCFGs)
  - Markov networks

Various Search/Optimization Algorithms

- Gradient descent
  - Perceptron
  - Backpropagation
- Dynamic Programming
  - HMM Learning
  - PCFG Learning
- Divide and Conquer
  - Decision tree induction
  - Rule learning
- Evolutionary Computation
  - Genetic Algorithms (GAs)
  - Genetic Programming (GP)
  - Neuro-evolution

Evaluation

- Accuracy
- Precision and recall
- Squared error
- Likelihood
- Posterior probability
- Cost / Utility
- Margin
- Entropy
- K-L divergence
- etc.

ML in Practice

Loop

- Understand domain, prior knowledge, and goals
- Data integration, selection, cleaning, pre-processing, etc.
- Learn models
- Interpret results
- Consolidate and deploy discovered knowledge

Lessons Learned about Learning

- Learning can be viewed as using direct or indirect experience to approximate a chosen target function.

- Function approximation can be viewed as a search through a space of hypotheses (representations of functions) for one that best fits a set of training data.

- Different learning methods assume different hypothesis spaces (representation languages) and/or employ different search techniques.

Framework of ML

Training data: $\{(\boldsymbol{x}^{\mathbf{1}}, \hat{y}^1), (\boldsymbol{x}^{\mathbf{2}}, \hat{y}^2), \dots, (\boldsymbol{x}^{\boldsymbol{N}}, \hat{y}^N)\}$

Training:

Step 1: function with unknown → Step 2: define loss from training data → Step 3: optimization

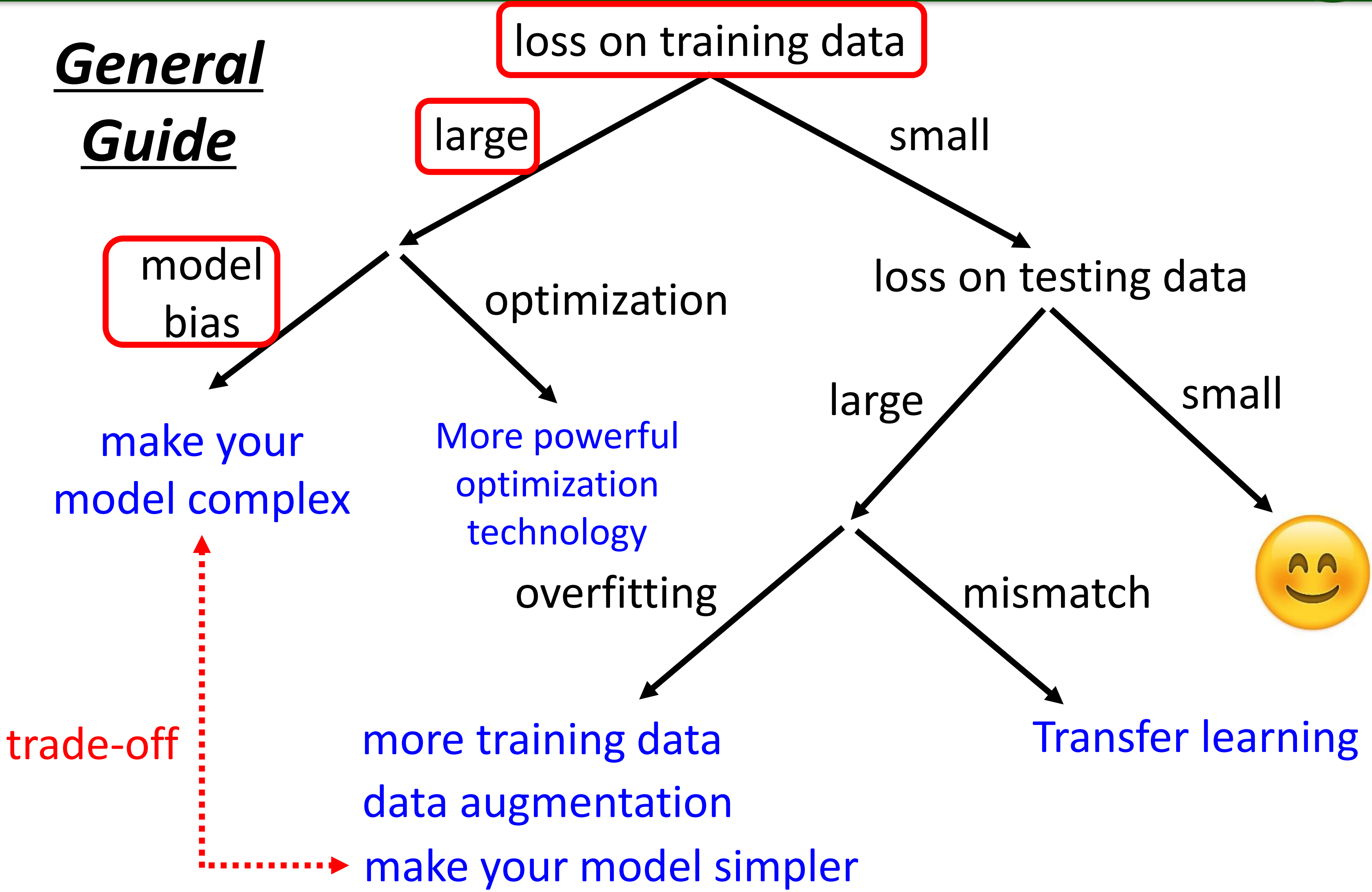
$y = f_{\boldsymbol{\theta}}(\boldsymbol{x})$ $\quad L(\boldsymbol{\theta})$ $\quad \boldsymbol{\theta}^* = arg \min_{\boldsymbol{\theta}} L$

Testing data: $\{\boldsymbol{x}^{\boldsymbol{N+1}}, \boldsymbol{x}^{\boldsymbol{N+2}}, \dots, \boldsymbol{x}^{\boldsymbol{N+M}}\}$

Use $y = f_{\boldsymbol{\theta}^*}(\boldsymbol{x})$ to label the testing data

$\{y^{N+1}, y^{N+2}, \dots, y^{N+M}\}$

Framework of ML



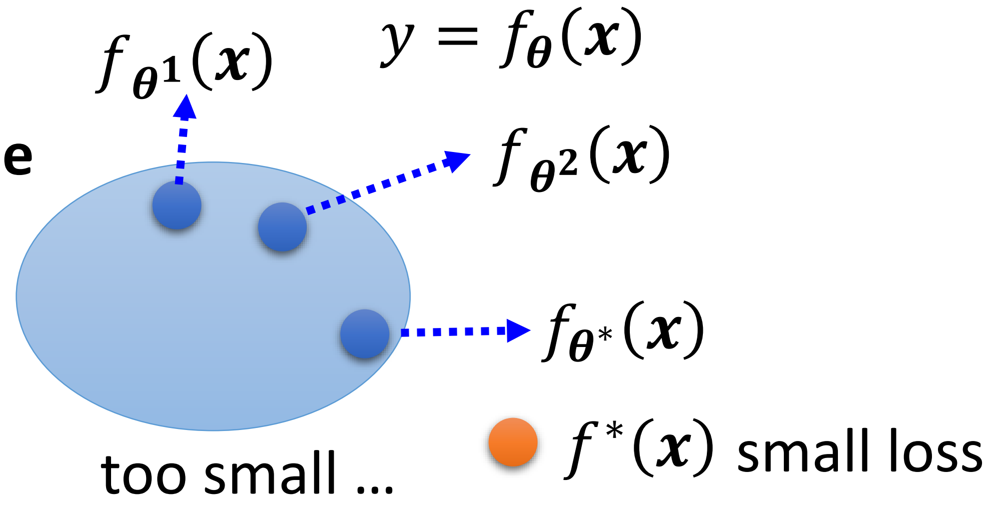
Framework of ML —— Model Bias

Model Bias

- **The model is too simple**

find a needle in a haystack …

… but there is no needle



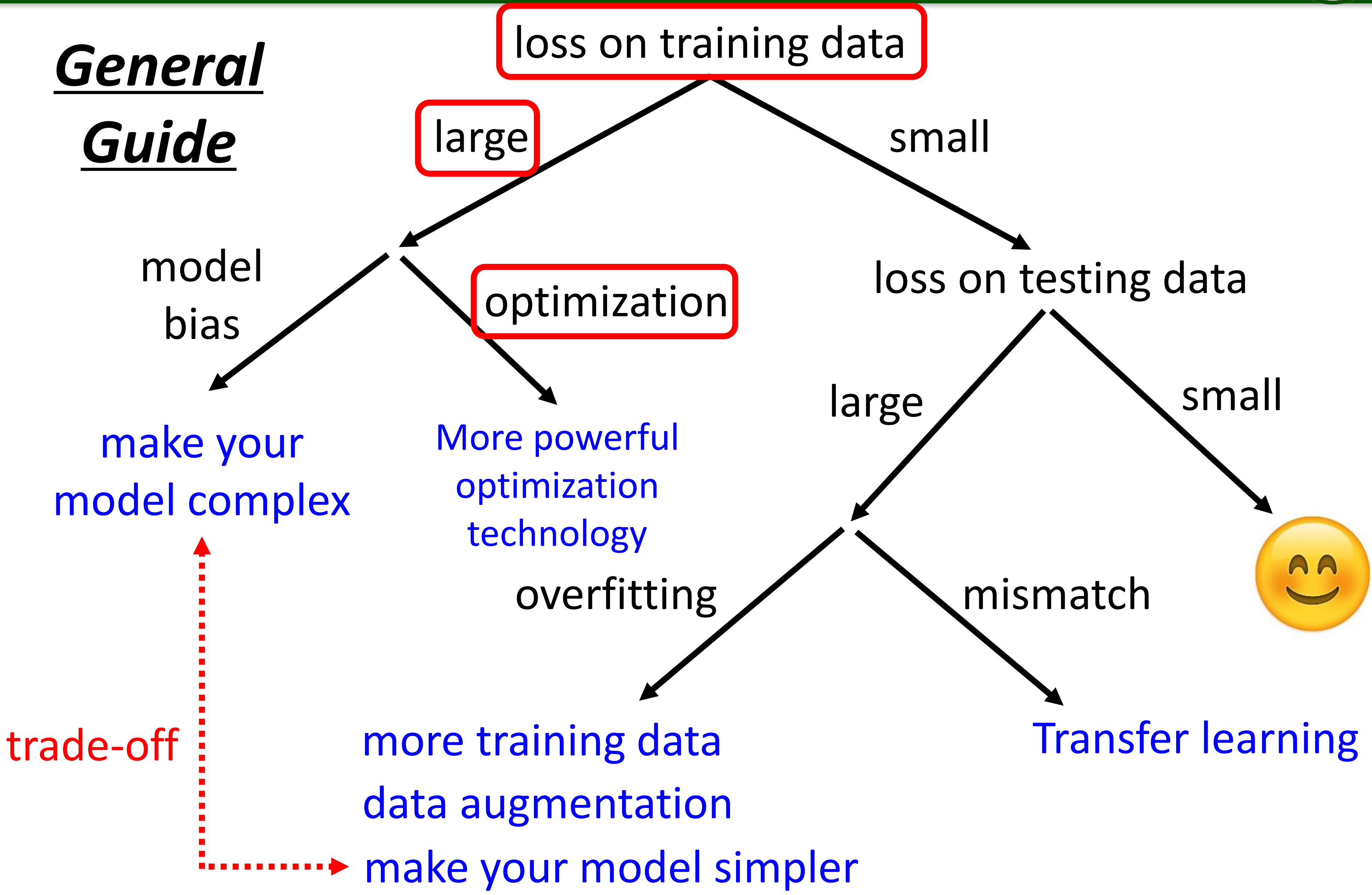
- **Solution: redesign your model to make it more flexible**

$$y = b + w x_1 \xrightarrow{\text{More features}} y = b + \sum_{j=1}^{56} w_j x_j$$

Deep Learning
(more neurons, layers)

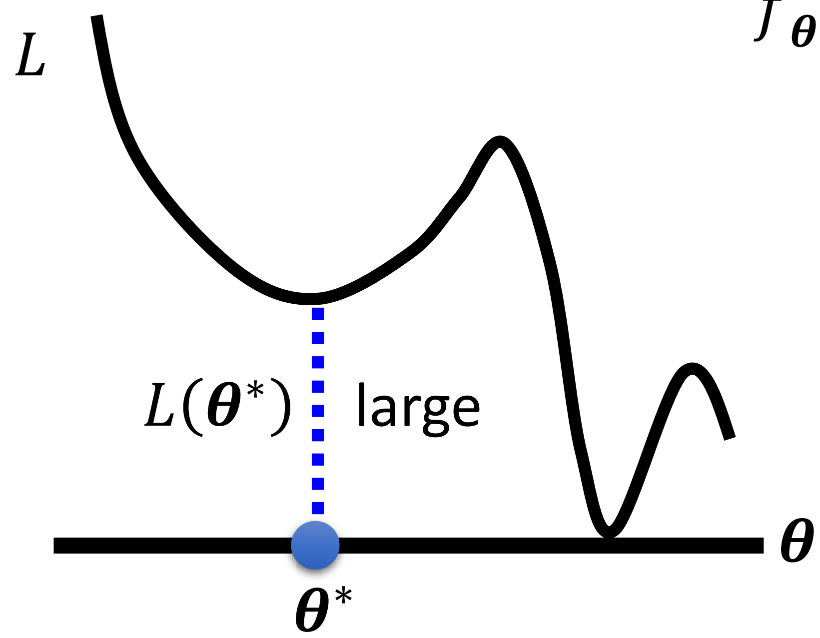
$$y = b + \sum_i c_i \, sigmoid\left(b_i + \sum_j w_{ij} x_j \right)$$

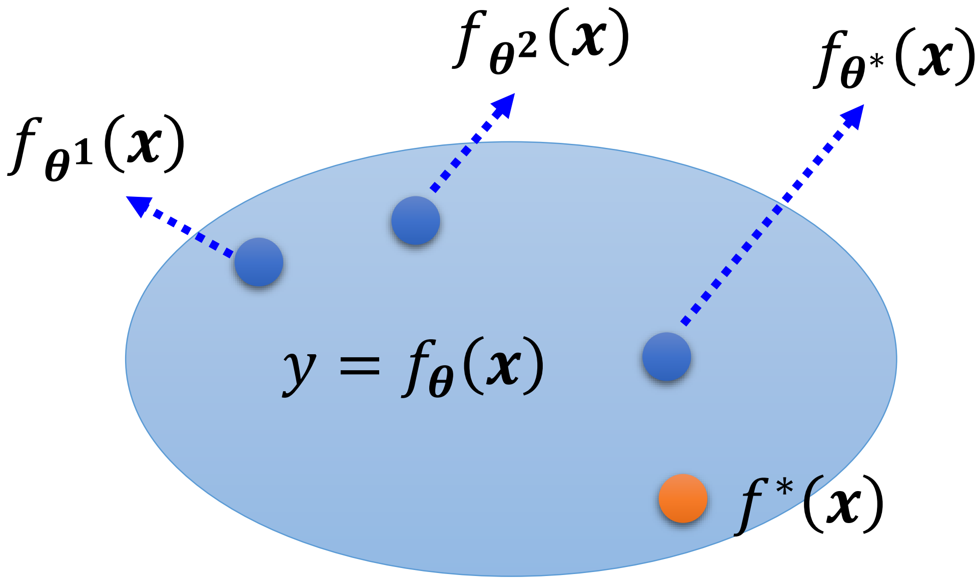
Framework of ML



Optimization Issue

- **Large loss not always imply model bias. There is another possibility ...**





A needle is in a haystack ...

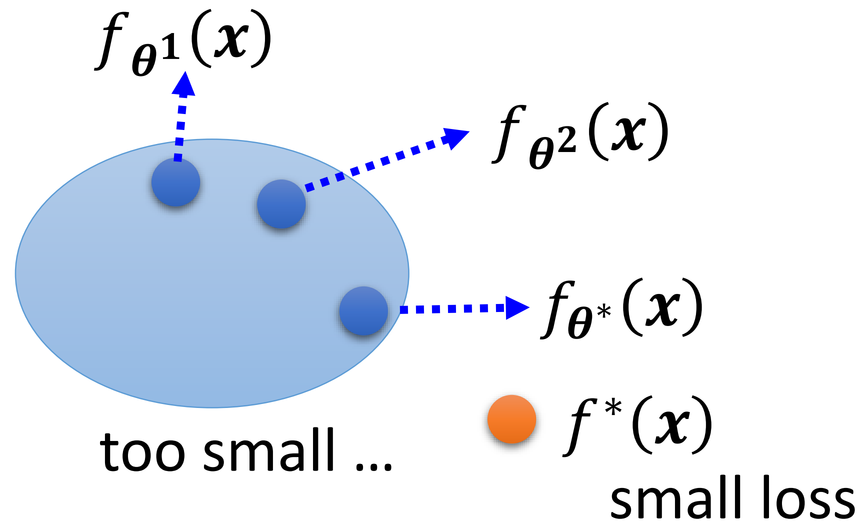
... Just cannot find it.

Framework of ML —— Optimization

Model Bias

find a needle in a haystack …

… but there is no needle

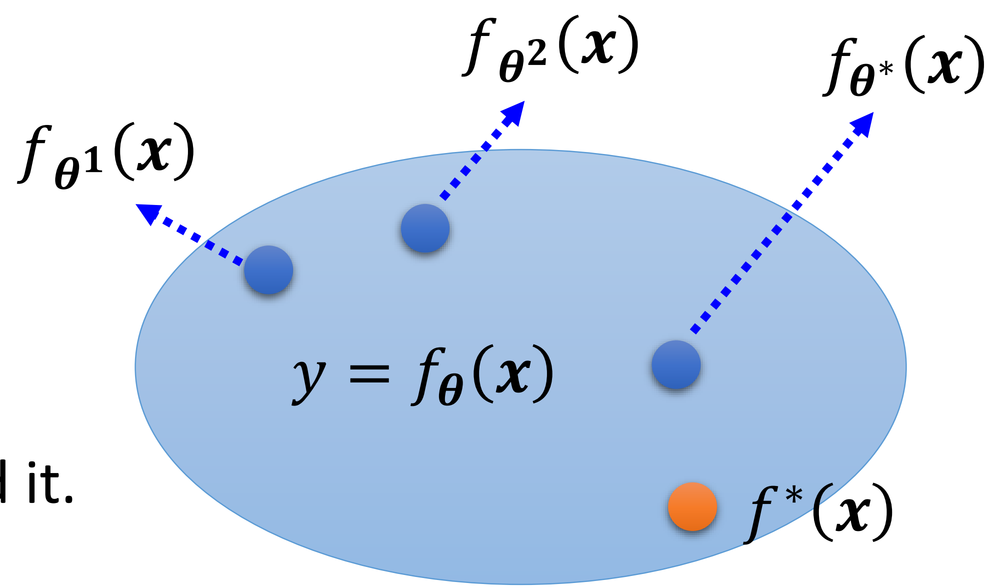


Which one???

Optimization Issue

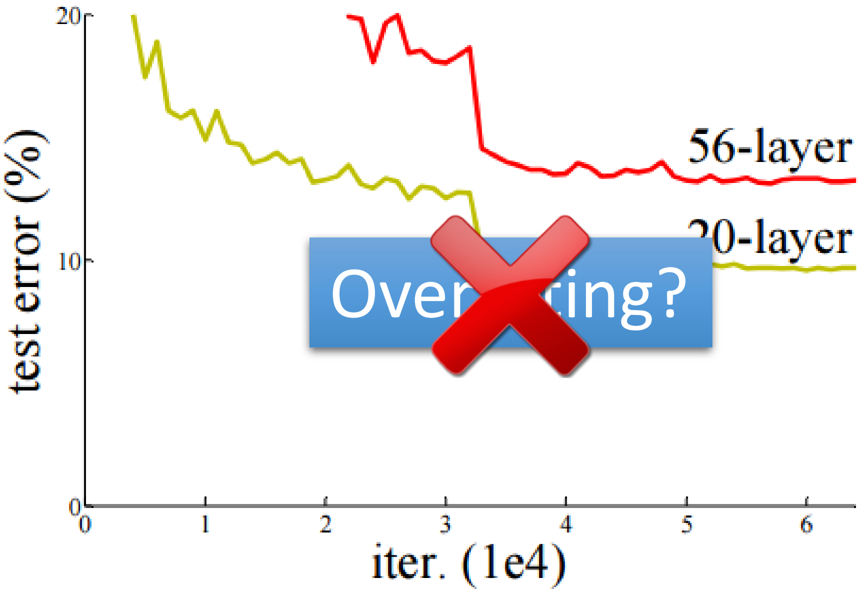
A needle is in a haystack …

… Just cannot find it.

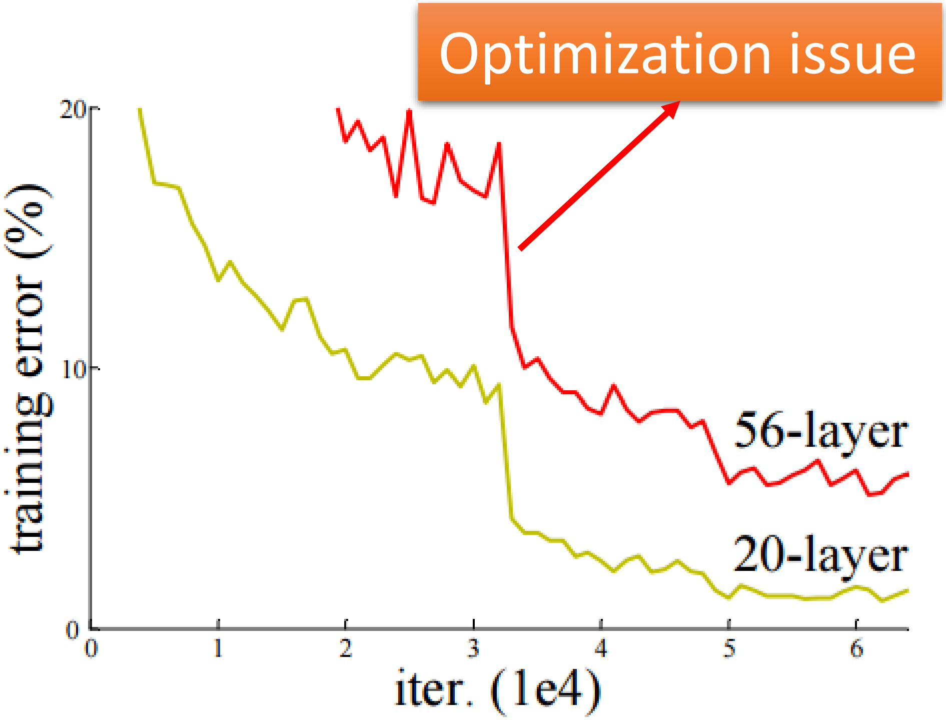


Model Bias v.s. Optimization Issue

- Gaining the insights from comparison



Testing Data



Training Data

Ref: http://arxiv.org/abs/1512.03385

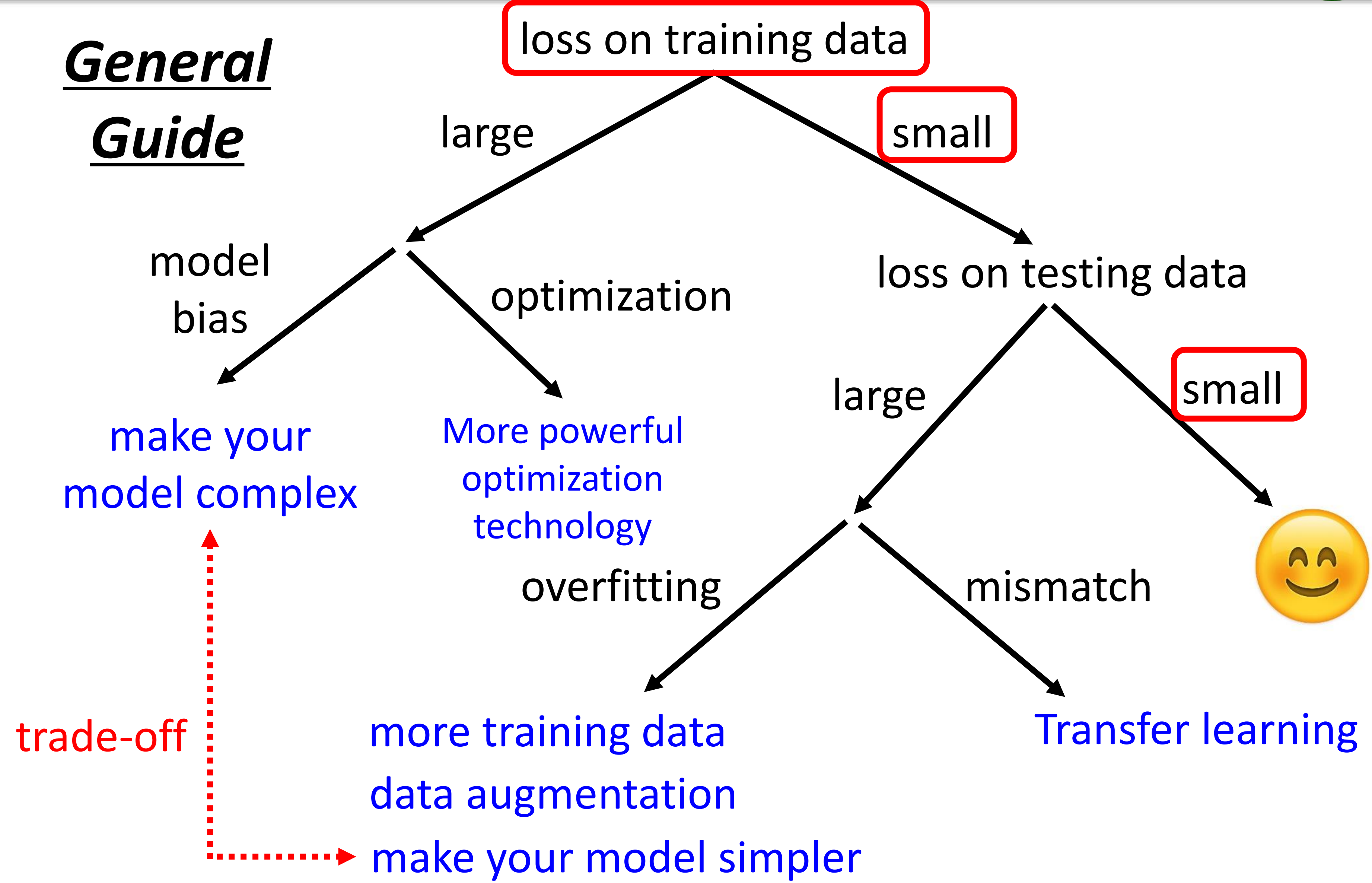
Optimization Issue

- Gaining the insights from comparison
- Start from shallower networks (or other models), which are easier to optimize.
- If deeper networks do not obtain smaller loss on **training data**, then there is optimization issue.

	1 layer	2 layer	3 layer	4 layer	5 layer
2017 – 2020	0.28k	0.18k	0.14k	0.10k	0.34k

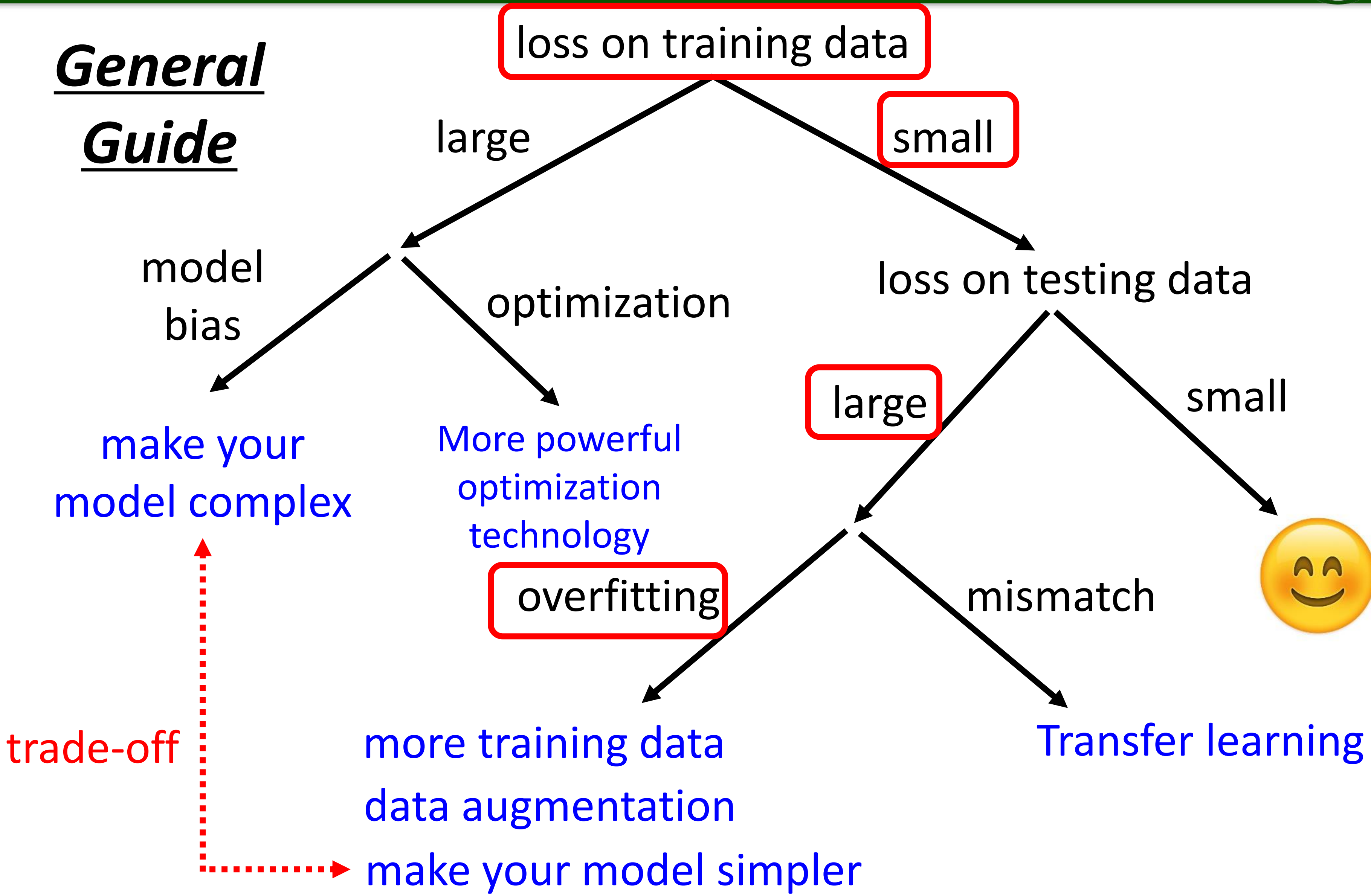
- Solution: More powerful optimization technology

Framework of ML



Framework of ML

General Guide



Framework of ML —— Overfitting

- Small loss on training data, large loss on testing data. Why?

<u>An extreme example</u>

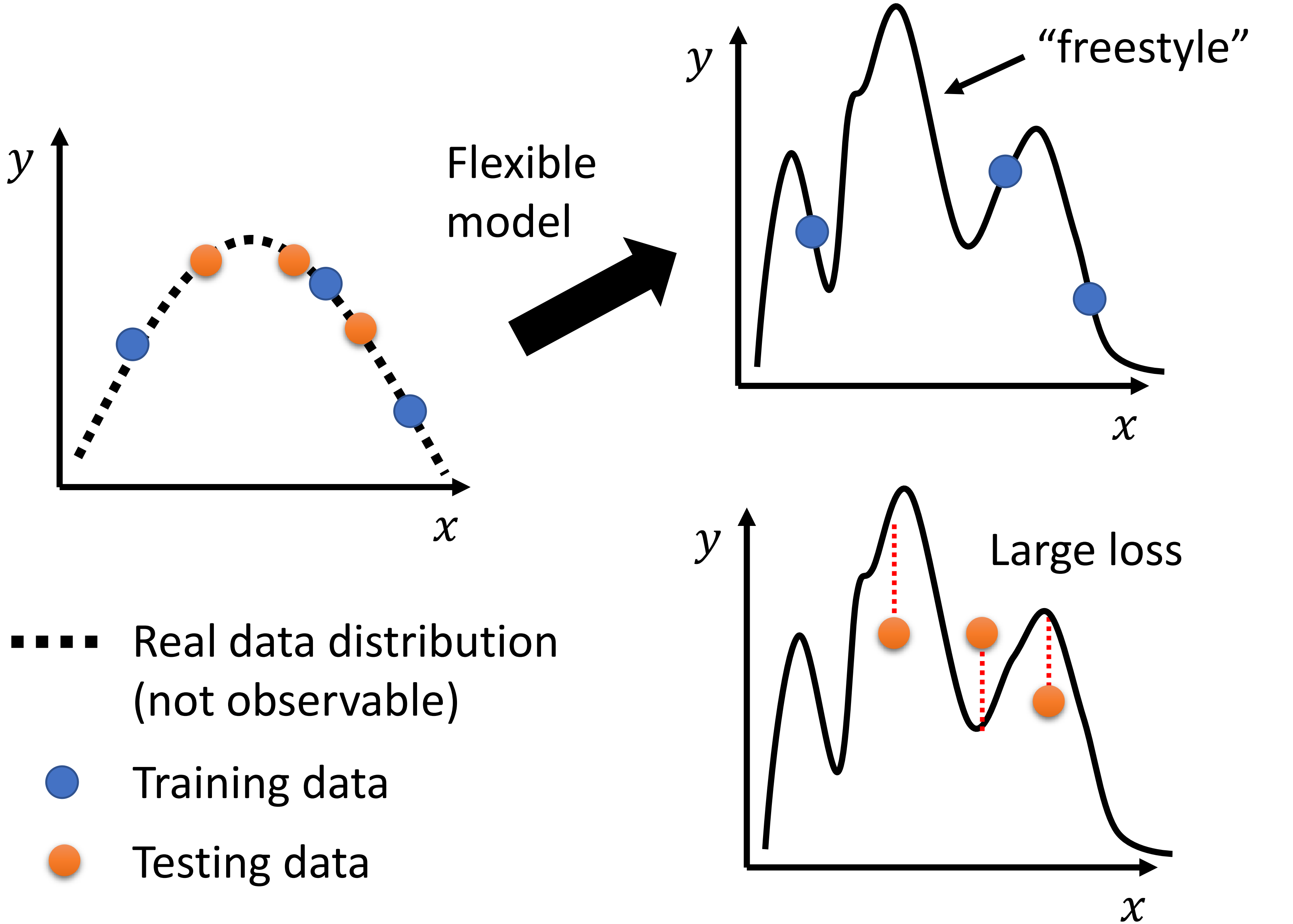
Training data: $\left\{\left(\boldsymbol{x}^{\mathbf{1}}, \hat{y}^1\right), \left(\boldsymbol{x}^{\mathbf{2}}, \hat{y}^2\right), \ldots, \left(\boldsymbol{x}^{\boldsymbol{N}}, \hat{y}^N\right)\right\}$

$$f(\boldsymbol{x}) = \begin{cases} \hat{y}^i & \exists \boldsymbol{x}^{\boldsymbol{i}} = \boldsymbol{x} \\ random & otherwise \end{cases}$$

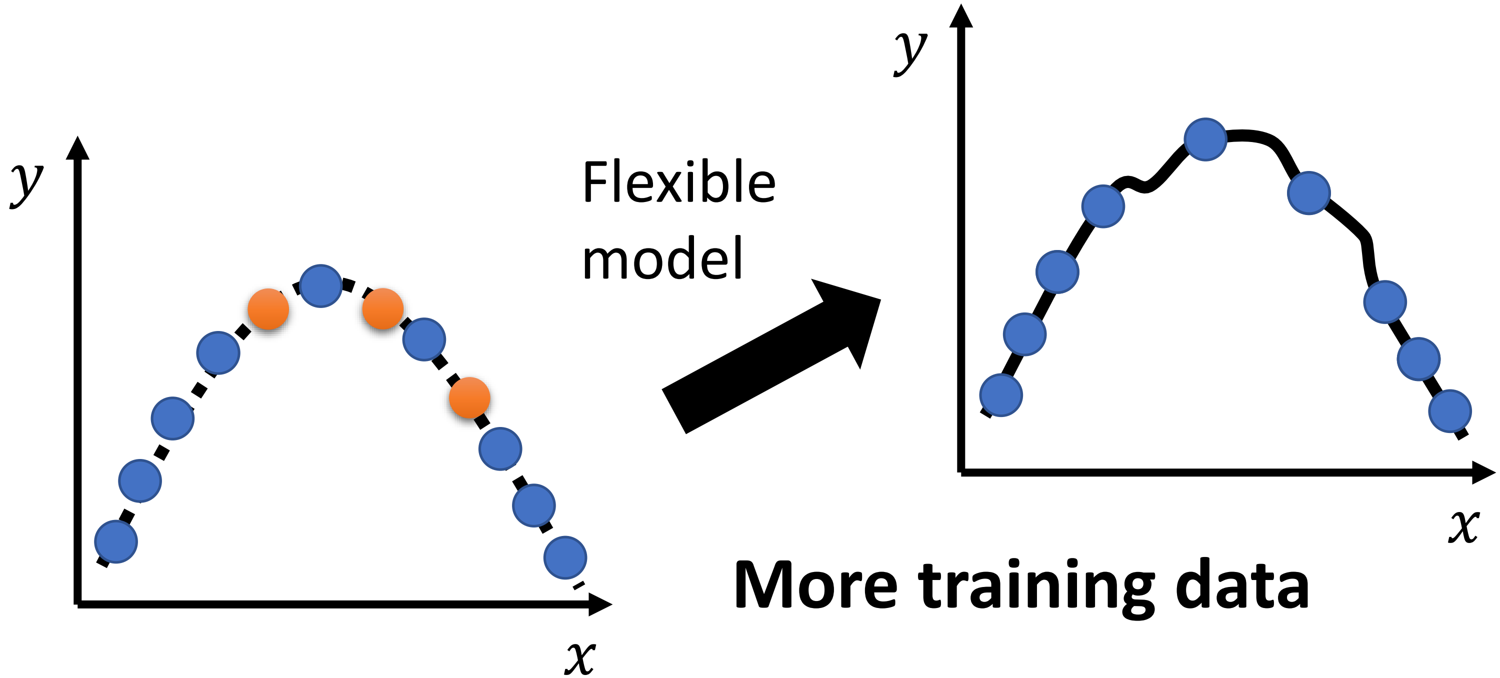
Less than useless …

This function obtains **zero training loss**, but **large testing loss**.

Framework of ML —— Overfitting



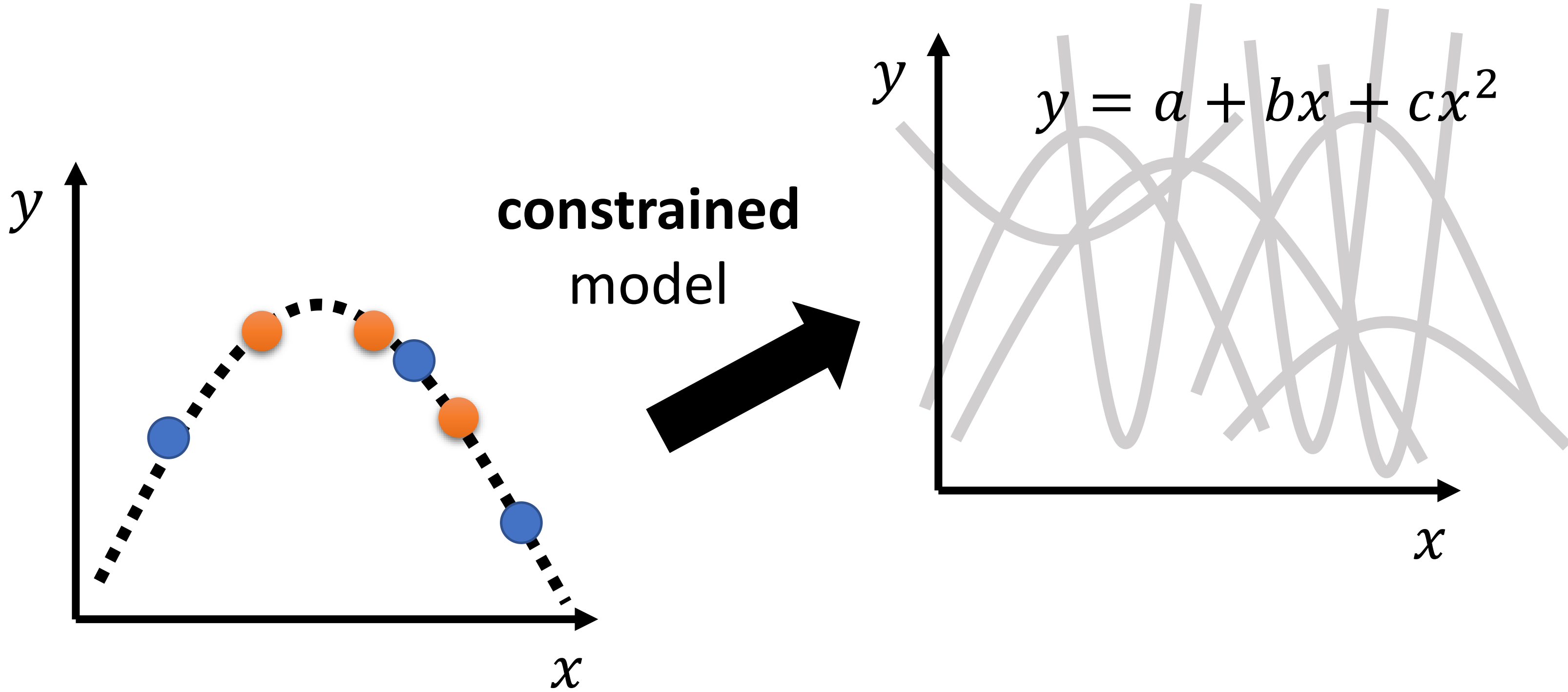
Framework of ML —— Overfitting



More training data

Data augmentation

Framework of ML —— Overfitting

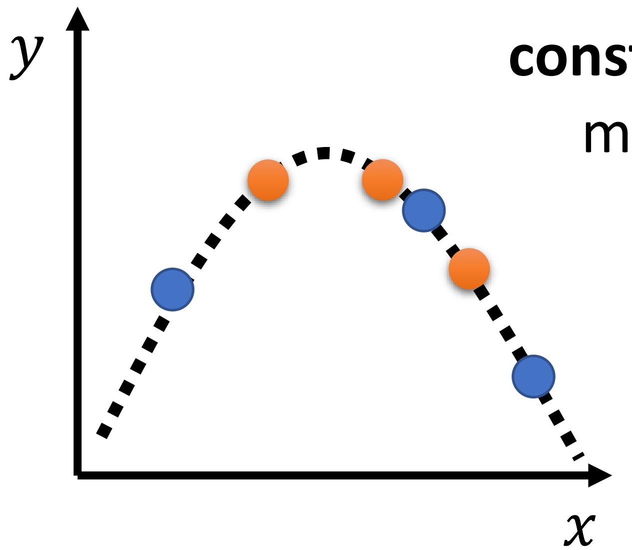


Real data distribution (not observable)

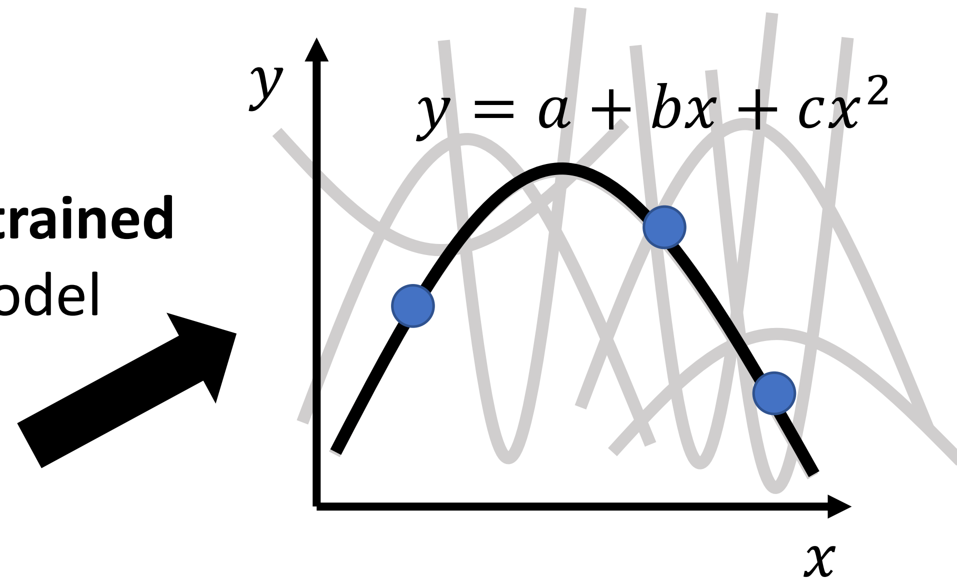
Training data

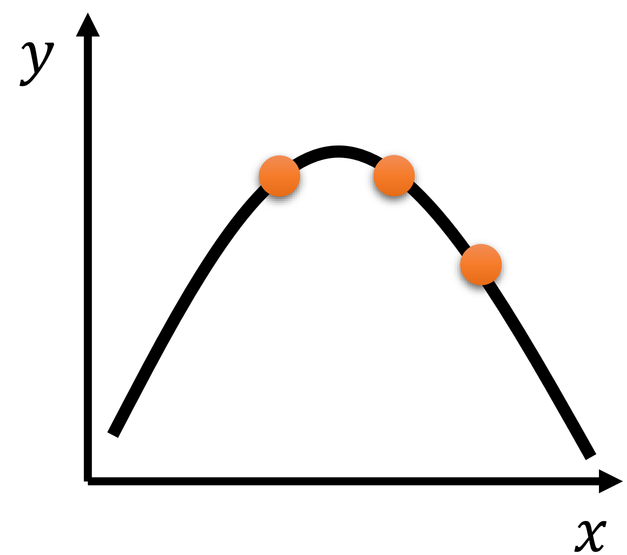
Testing data

Framework of ML —— Overfitting



constrained model



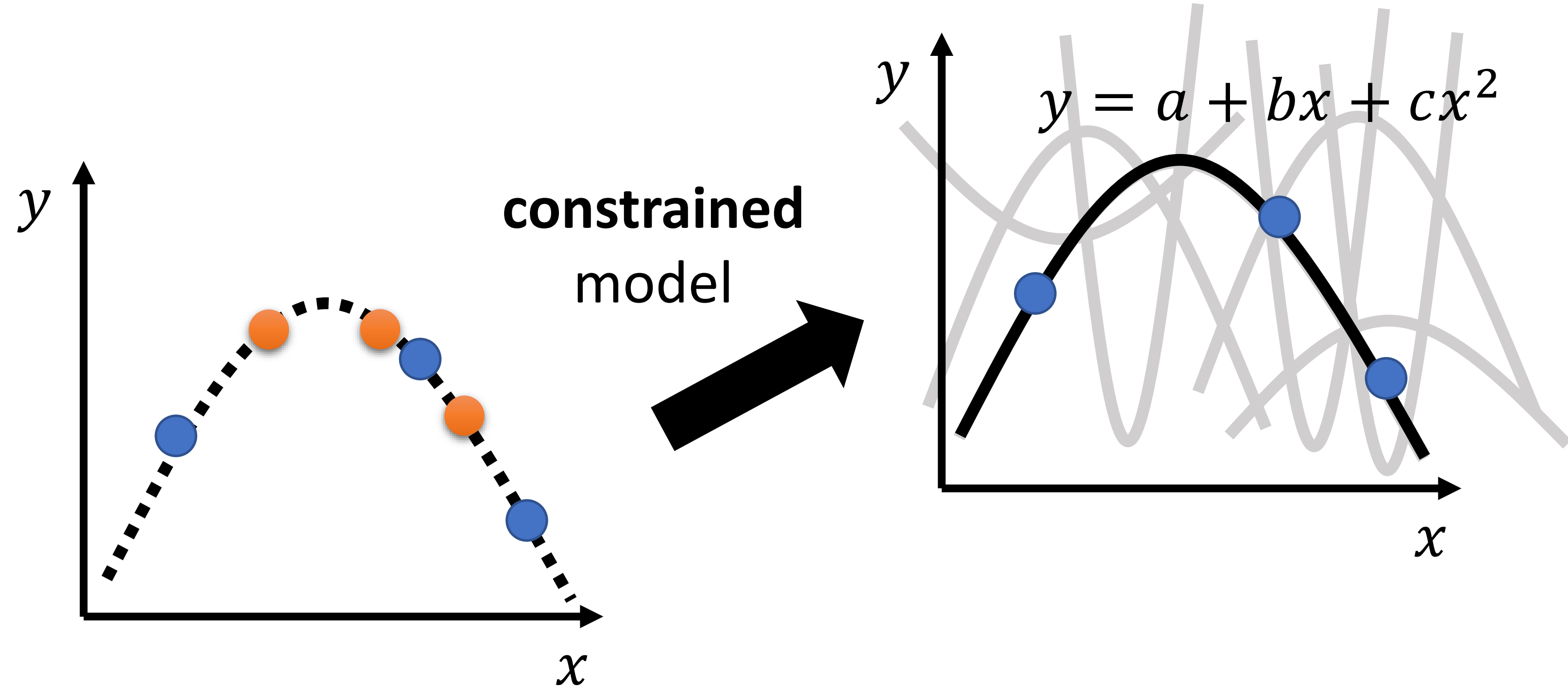


Real data distribution (not observable)

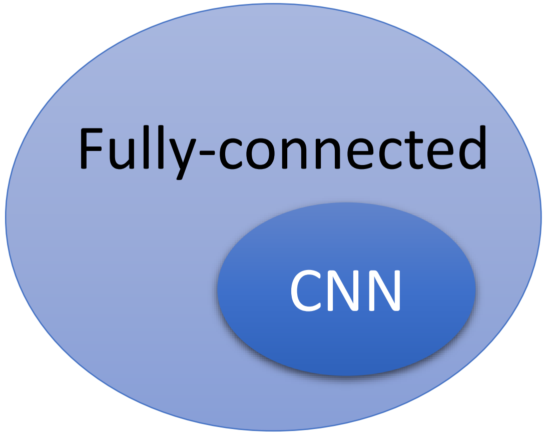
Training data

Testing data

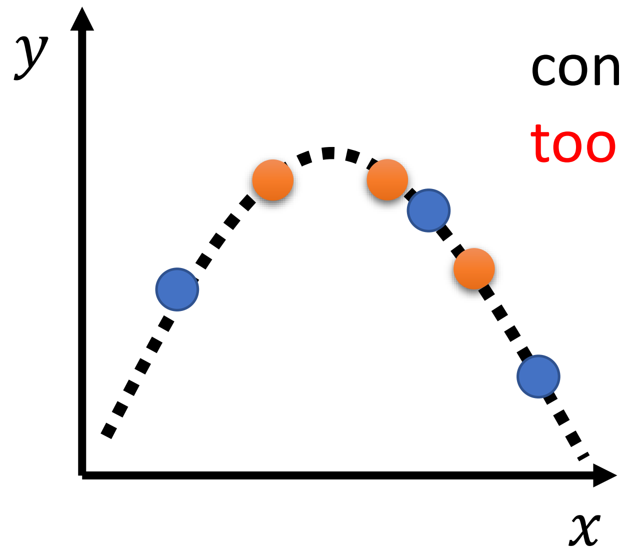
Framework of ML —— Overfitting



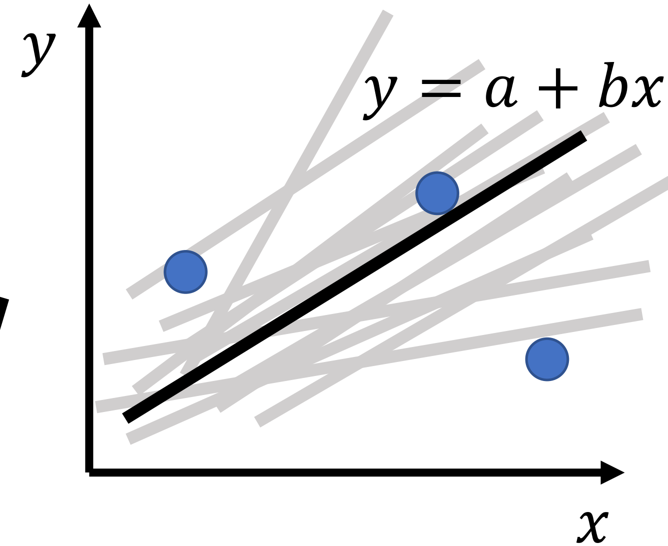
- Less parameters, sharing parameters
- Less features
- Early stopping
- Regularization
- Dropout



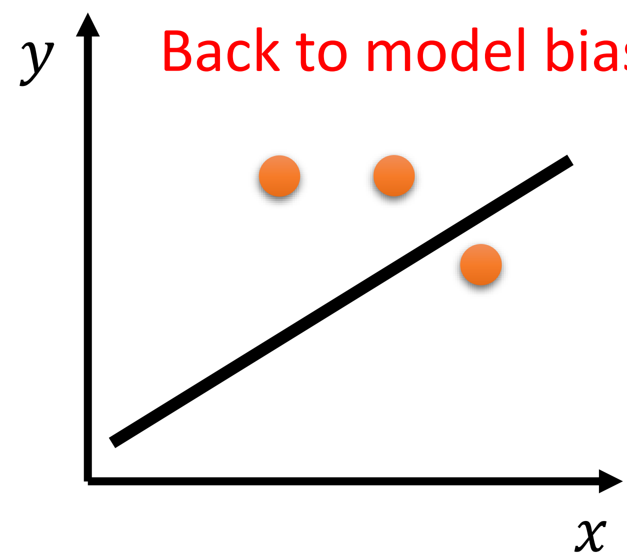
Framework of ML —— Overfitting



constrain
too much



Back to model bias …

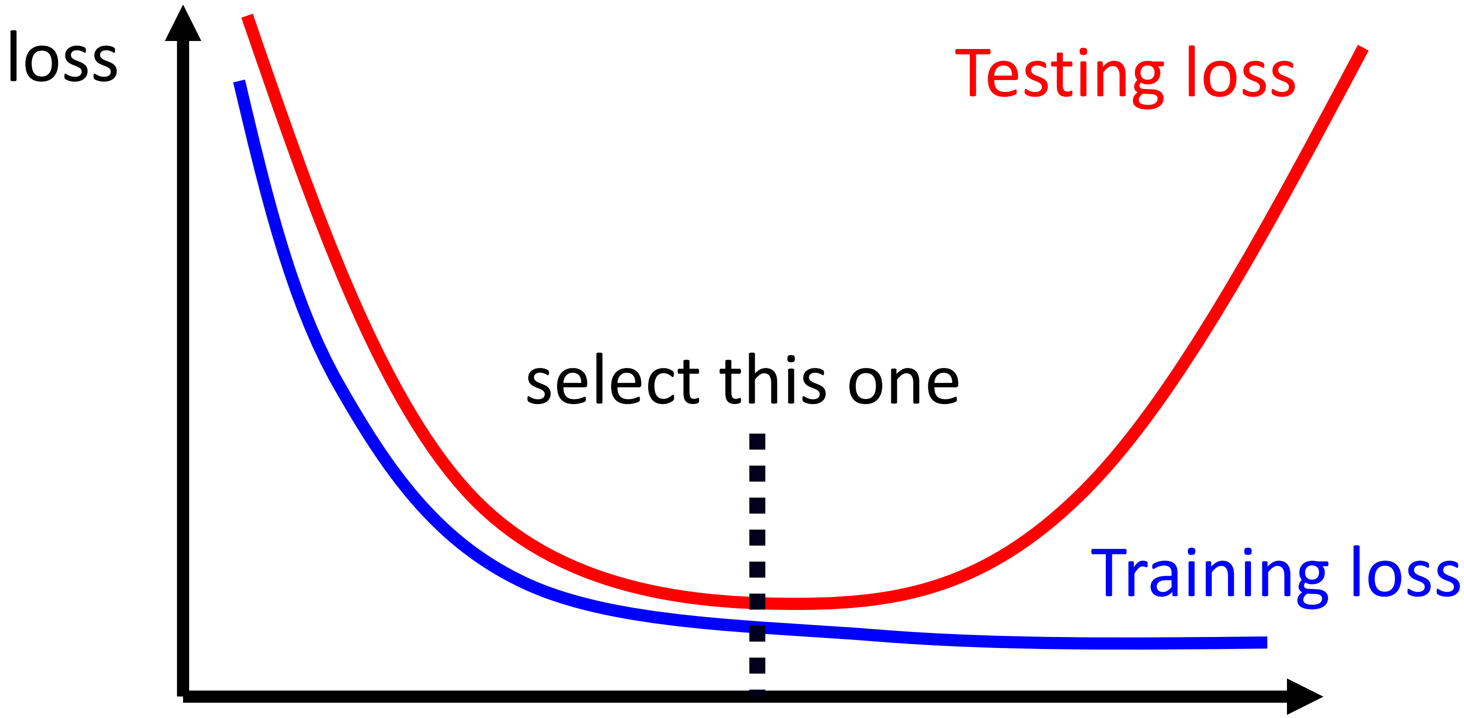


Real data distribution (not observable)

Training data

Testing data

Bias-Complexity Trade-off



Model becomes complex
(e.g. more features, more parameters)

Cross Validation

How to split?

Training Set

Training Set | Validation set

Model 1 → mse = 0.9

Model 2 → mse = 0.7

Model 3 → mse = 0.5 → mse > 0.5 → mse > 0.5

public

Testing Set

private

Testing Set

Using the results of public testing data to select your model

You are making public set better than private set.

Not recommend

Framework of ML —— Overfitting

N-fold Cross Validation

Training Set

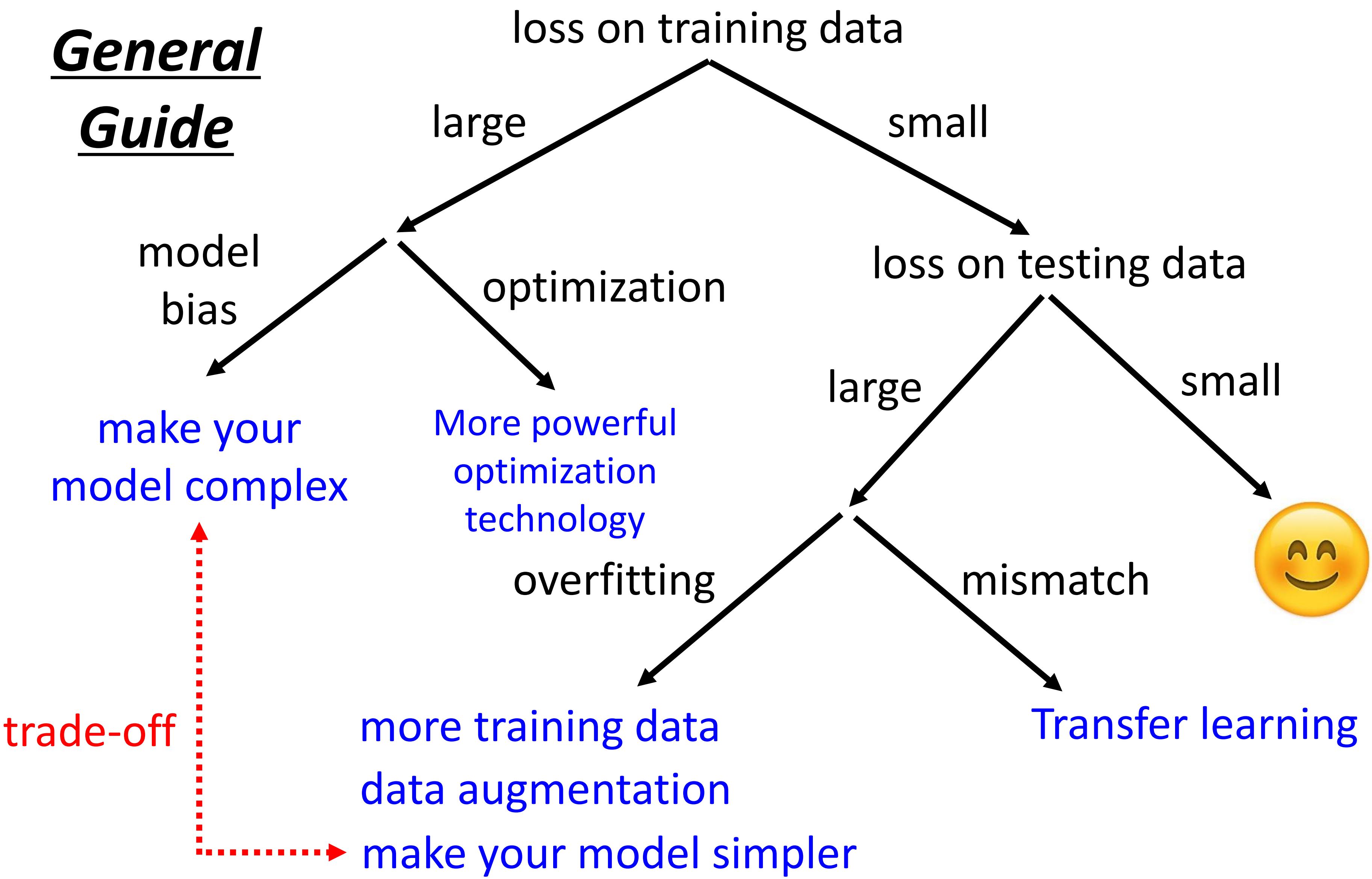
			Model 1	Model 2	Model 3
Train	Train	Val	mse = 0.2	mse = 0.4	mse = 0.4
Train	Val	Train	mse = 0.4	mse = 0.5	mse = 0.5
Val	Train	Train	mse = 0.3	mse = 0.6	mse = 0.3
			Avg mse = 0.3	Avg mse = 0.5	Avg mse = 0.4

Testing Set
public

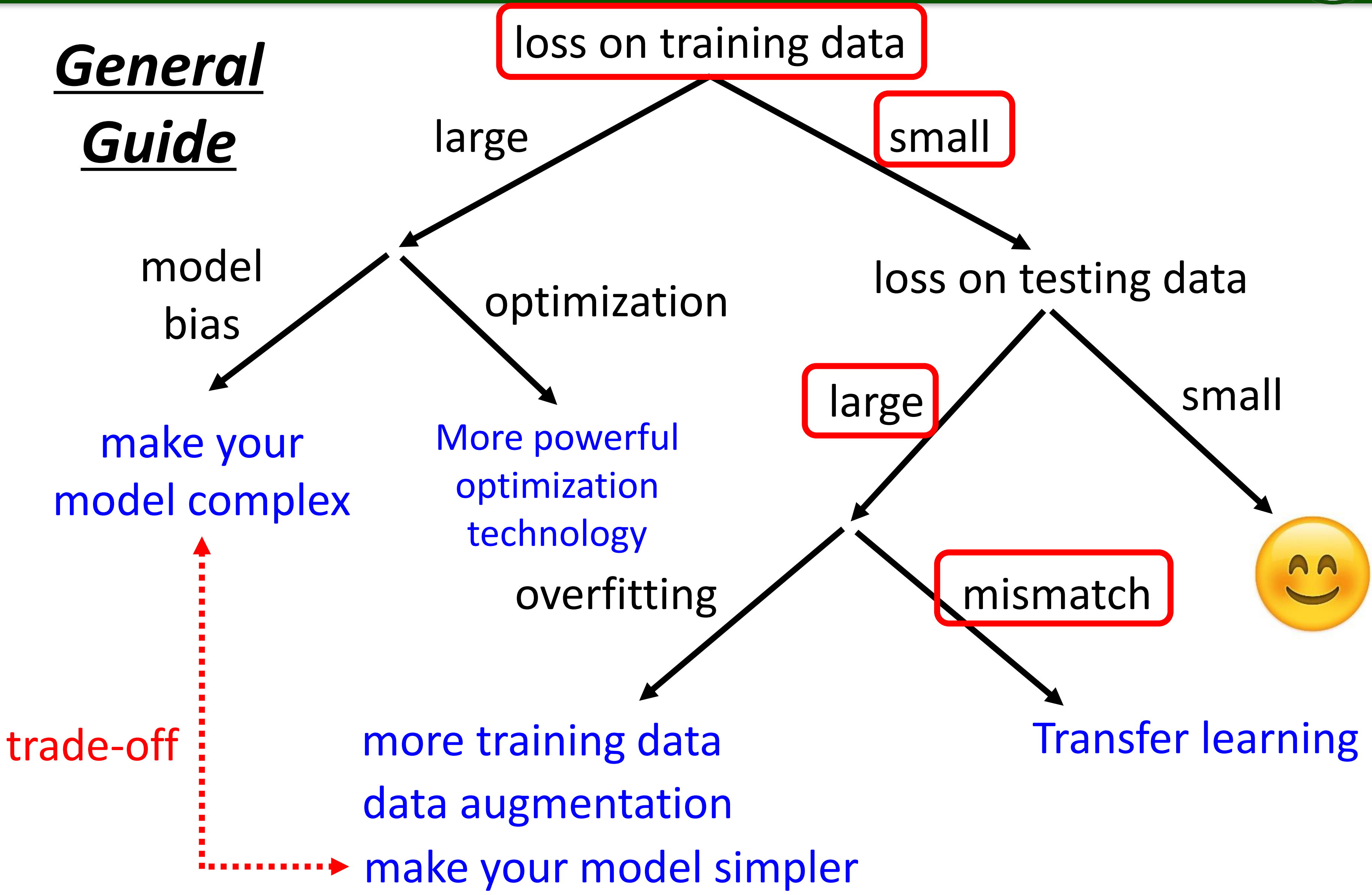
Testing Set
private

Framework of ML

General Guide



Framework of ML



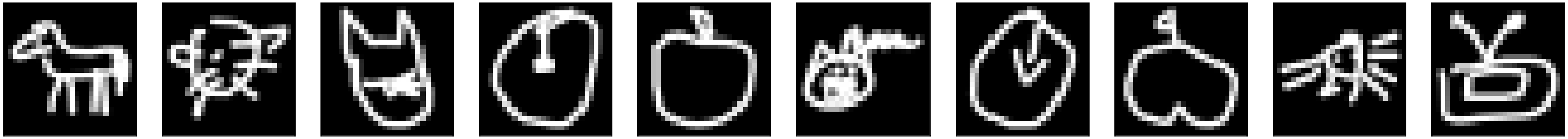
Framework of ML —— Mismatch

- Your training and testing data have different distributions. Be aware of how data is generated.

Training Data

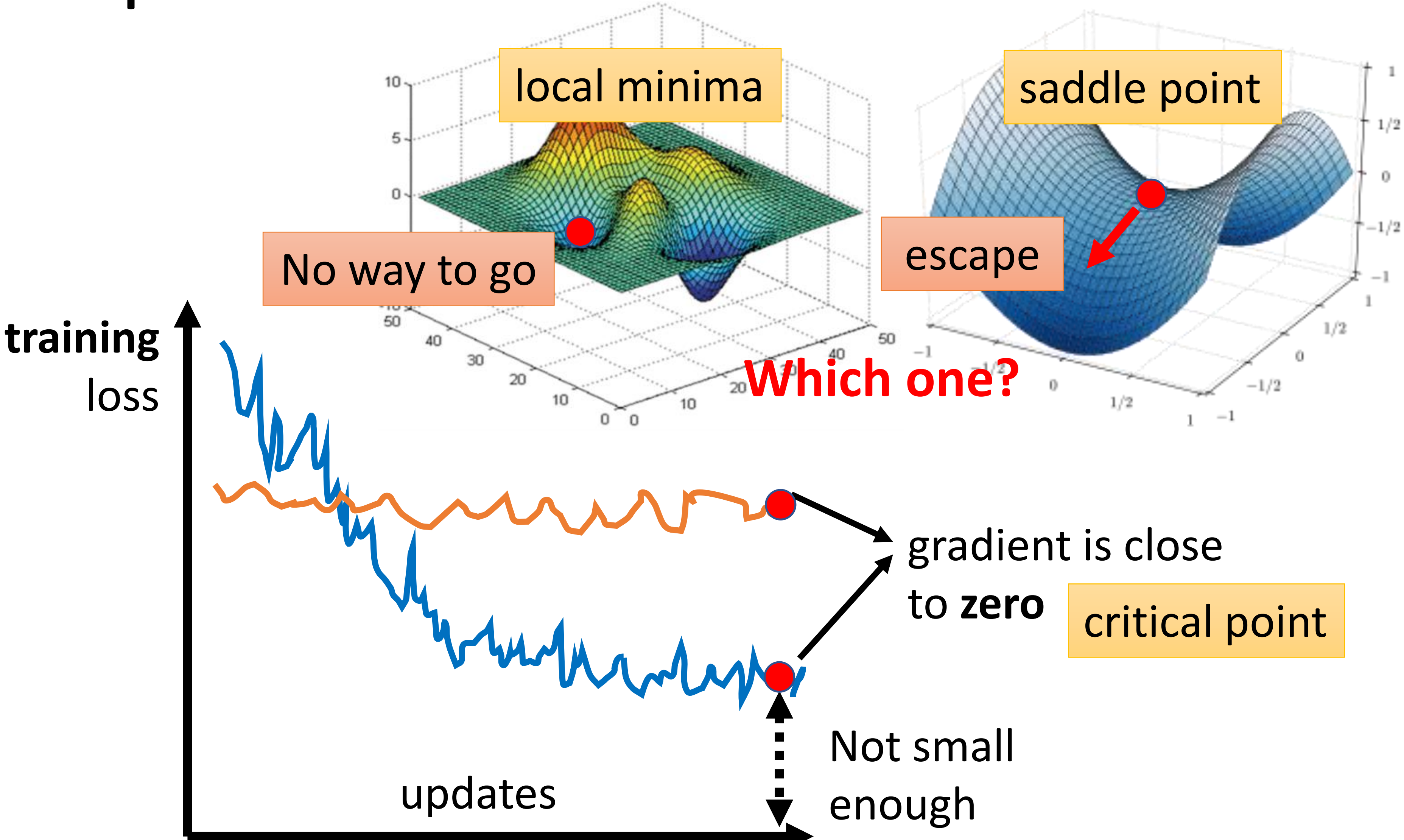
Simply increasing the training data will not help.

Testing Data



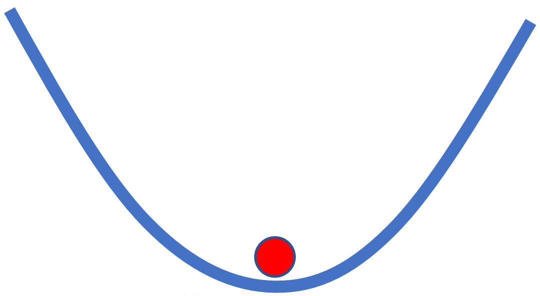
Small Gradient

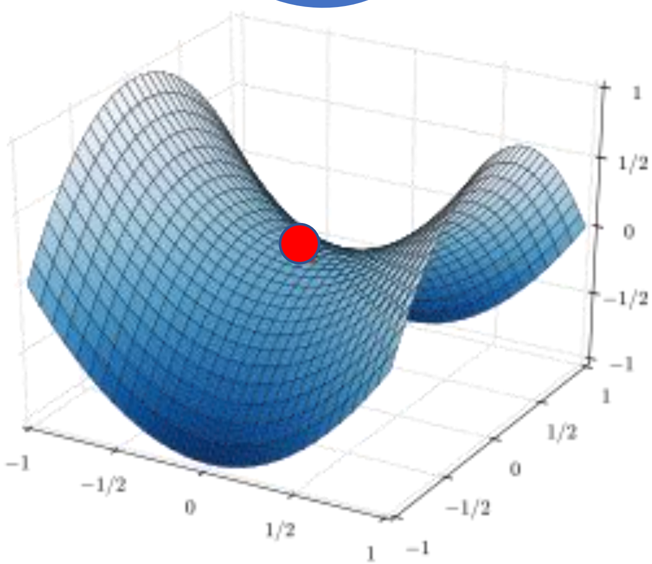
Optimization Fails because ……

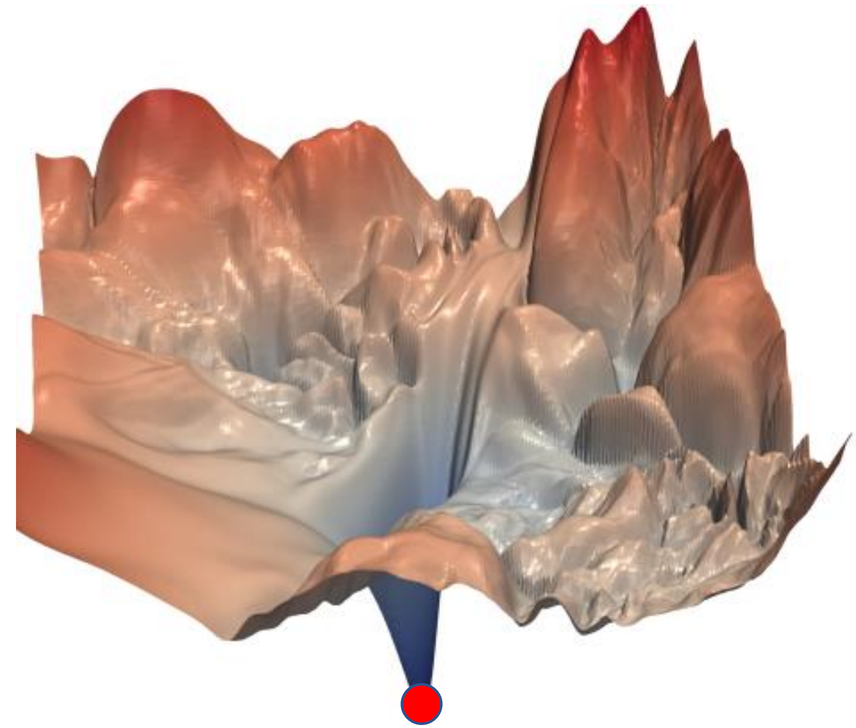


Small Gradient

Saddle Point v.s. Local Minima



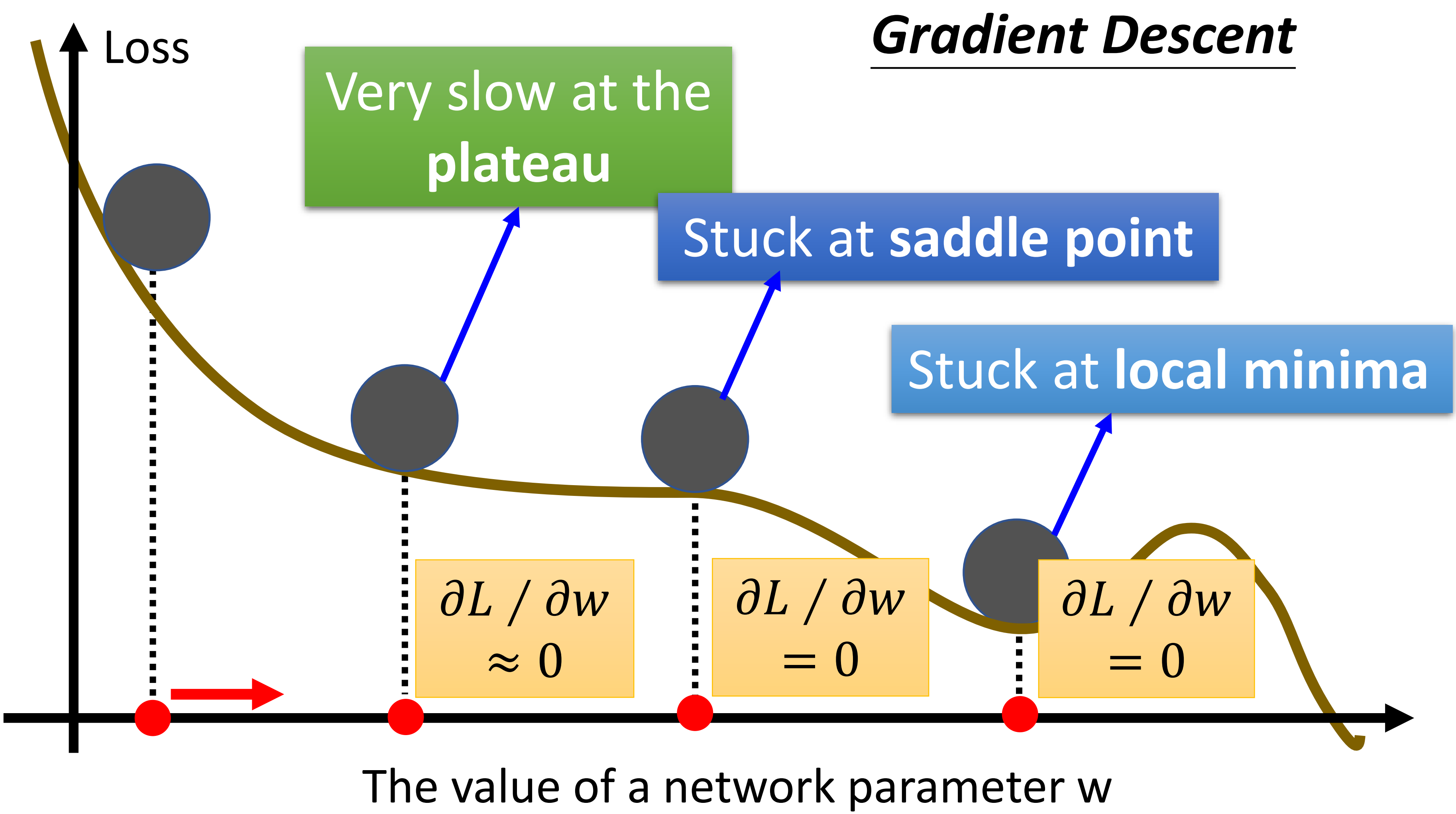




Saddle point in higher dimension?

When you have lots of parameters, perhaps local minima is rare?

Small Gradient

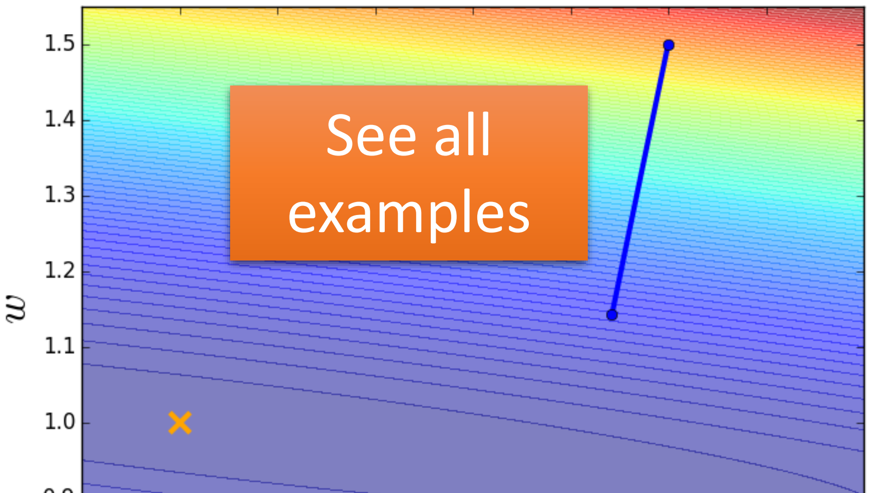


Small Batch v.s. Large Batch

Consider 20 examples (N=20)

Batch size = N (Full batch)

Update after seeing all the 20 examples

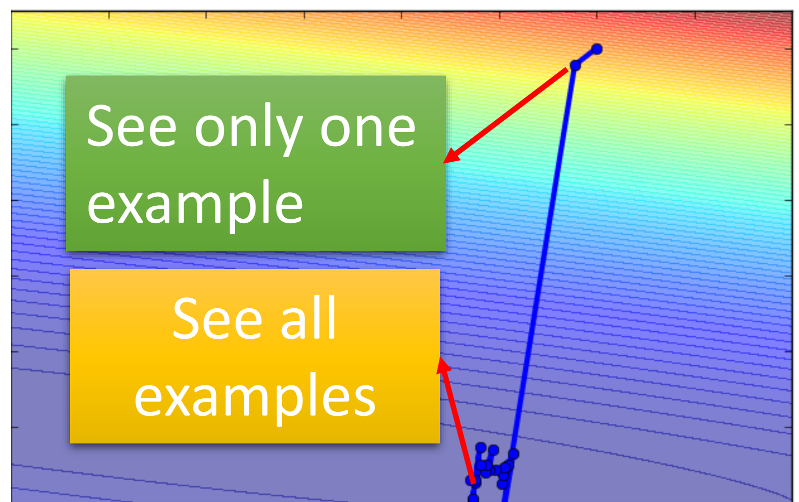


Long time for cooldown, but powerful

Batch size = 1

Update for each example

Update 20 times in an epoch



Short time for cooldown, but noisy

Small Batch v.s. Large Batch

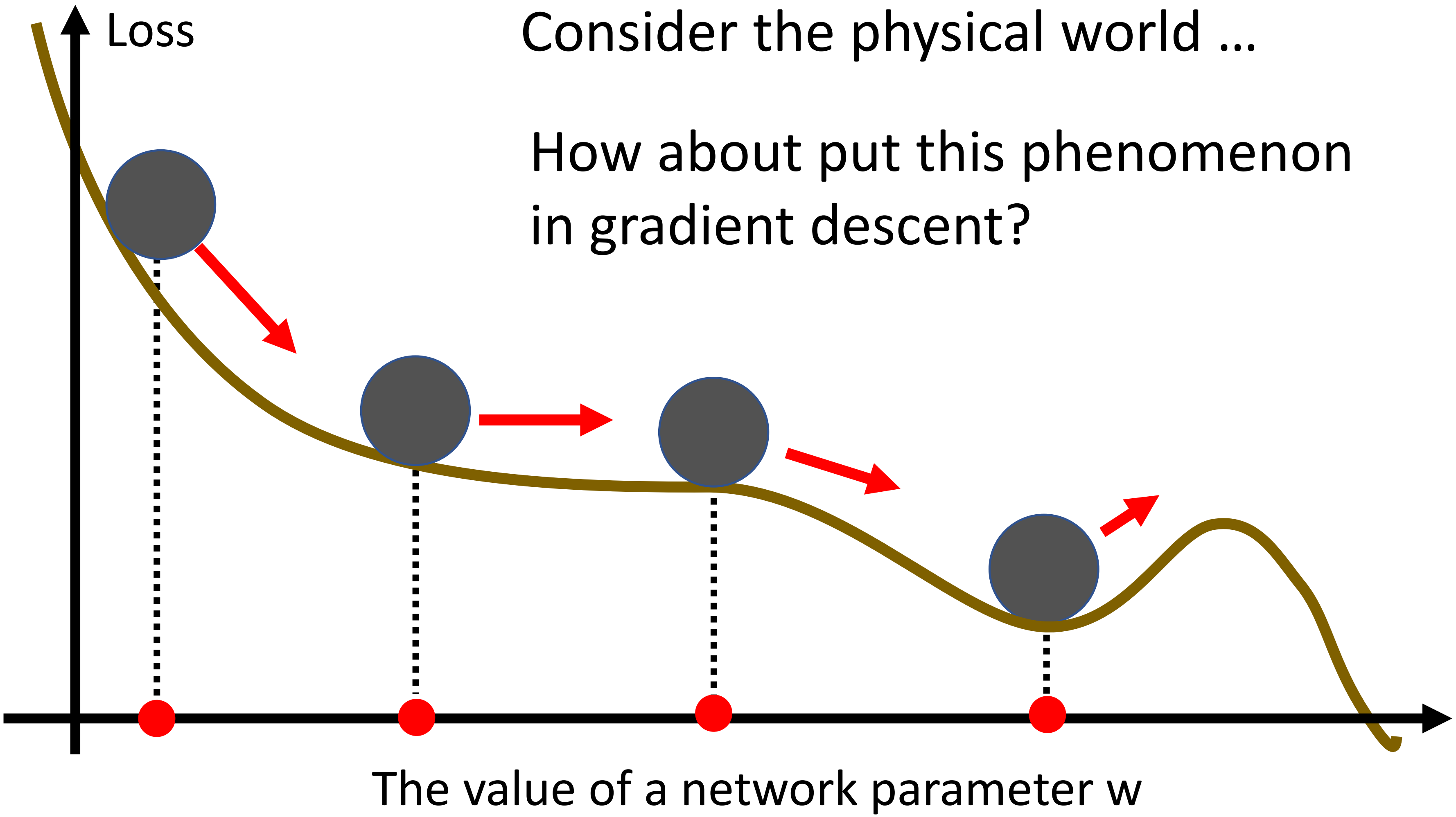
	Small	Large
Speed for one update (no parallel)	Faster	Slower
Speed for one update (with parallel)	Same	Same (not too large)
Time for one epoch	Slower	Faster WIN
Gradient	Noisy	Stable
Optimization	Better WIN	Worse
Generalization	Better WIN	Worse

Batch size is a hyperparameter you have to decide.

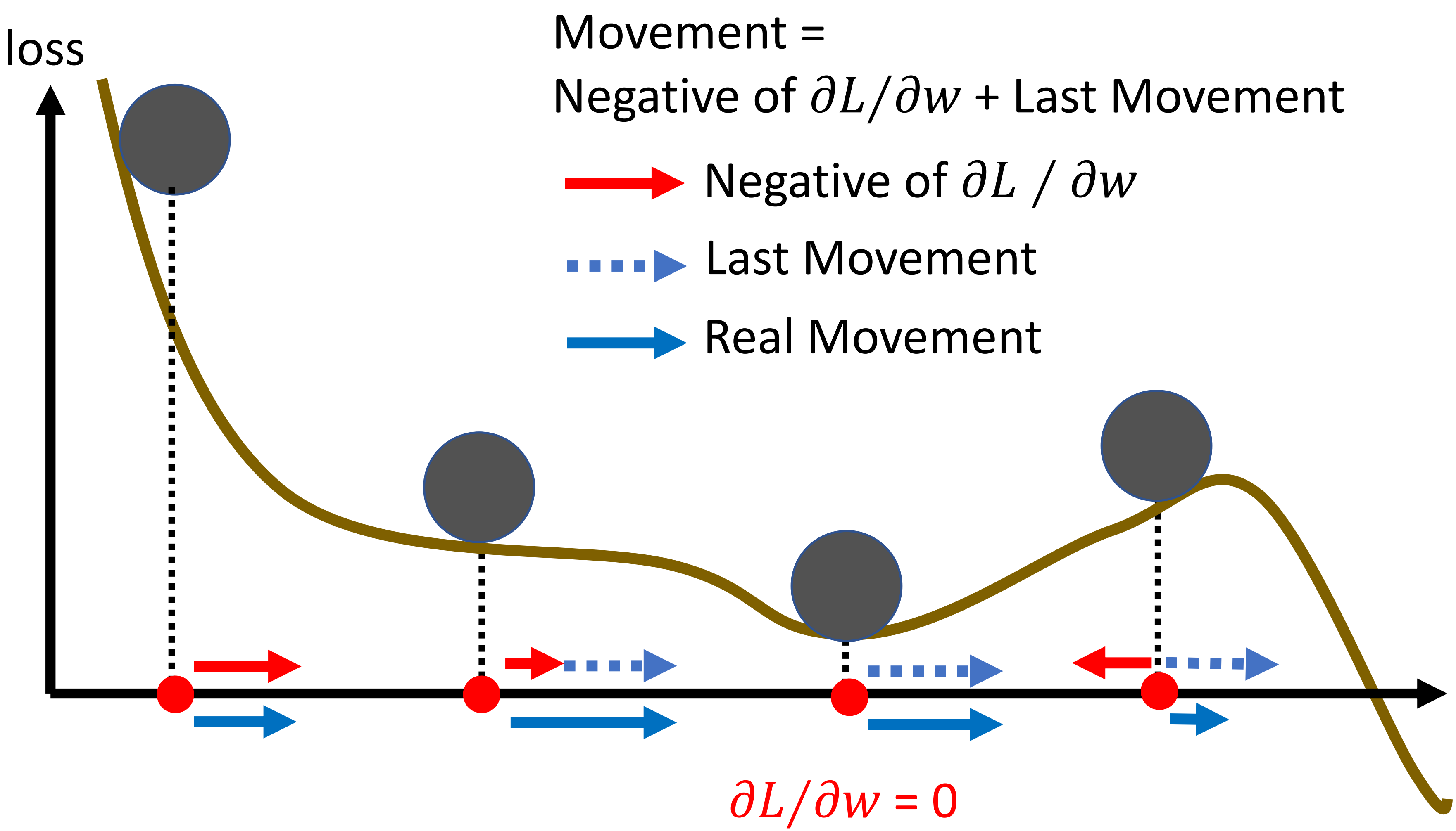
Small Gradient —— Tips2: Momentum

Consider the physical world …

How about put this phenomenon in gradient descent?

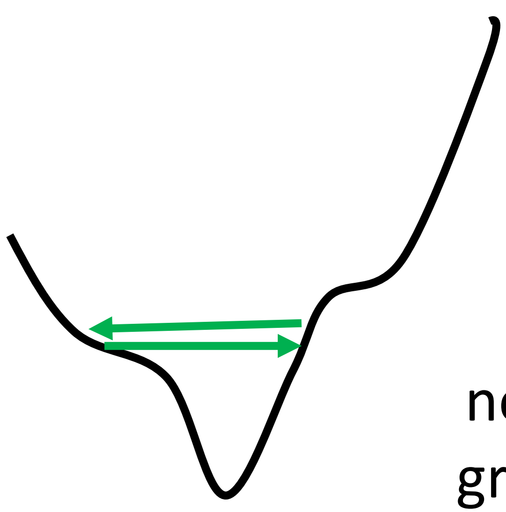


Gradient Descent + Momentum

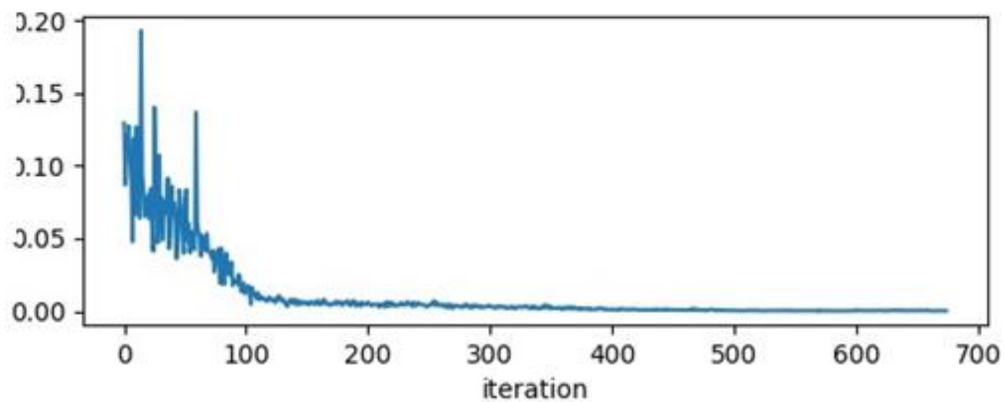


Optimizer

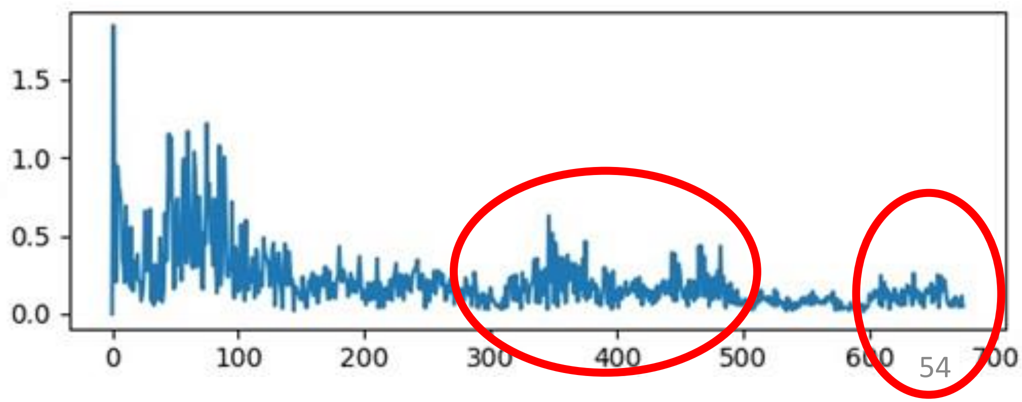
- Error surface is rugged: **Adaptive Learning Rate**
- People believe training stuck because the parameters are around a critical point …



loss



norm of gradient



Optimizer —— RMS

Root Mean Square

$$\boldsymbol{\theta}_i^{t+1} \leftarrow \boldsymbol{\theta}_i^{t} - \boxed{\frac{\eta}{\sigma_i^t}}\boldsymbol{g}_i^t$$

$$\boldsymbol{\theta}_i^{1} \leftarrow \boldsymbol{\theta}_i^{0} - \frac{\eta}{\sigma_i^0}\boldsymbol{g}_i^{\boldsymbol{0}} \qquad \sigma_i^0 = \sqrt{\left(\boldsymbol{g}_i^{\boldsymbol{0}}\right)^2} = \left|\boldsymbol{g}_i^{\boldsymbol{0}}\right|$$

$$\boldsymbol{\theta}_i^{2} \leftarrow \boldsymbol{\theta}_i^{1} - \frac{\eta}{\sigma_i^1}\boldsymbol{g}_i^{\boldsymbol{1}} \qquad \sigma_i^1 = \sqrt{\frac{1}{2}\left[\left(\boldsymbol{g}_i^{\boldsymbol{0}}\right)^2 + \left(\boldsymbol{g}_i^{\boldsymbol{1}}\right)^2\right]}$$

$$\boldsymbol{\theta}_i^{3} \leftarrow \boldsymbol{\theta}_i^{2} - \frac{\eta}{\sigma_i^2}\boldsymbol{g}_i^{\boldsymbol{2}} \qquad \sigma_i^2 = \sqrt{\frac{1}{3}\left[\left(\boldsymbol{g}_i^{\boldsymbol{0}}\right)^2 + \left(\boldsymbol{g}_i^{\boldsymbol{1}}\right)^2 + \left(\boldsymbol{g}_i^{\boldsymbol{2}}\right)^2\right]}$$

$$\vdots$$

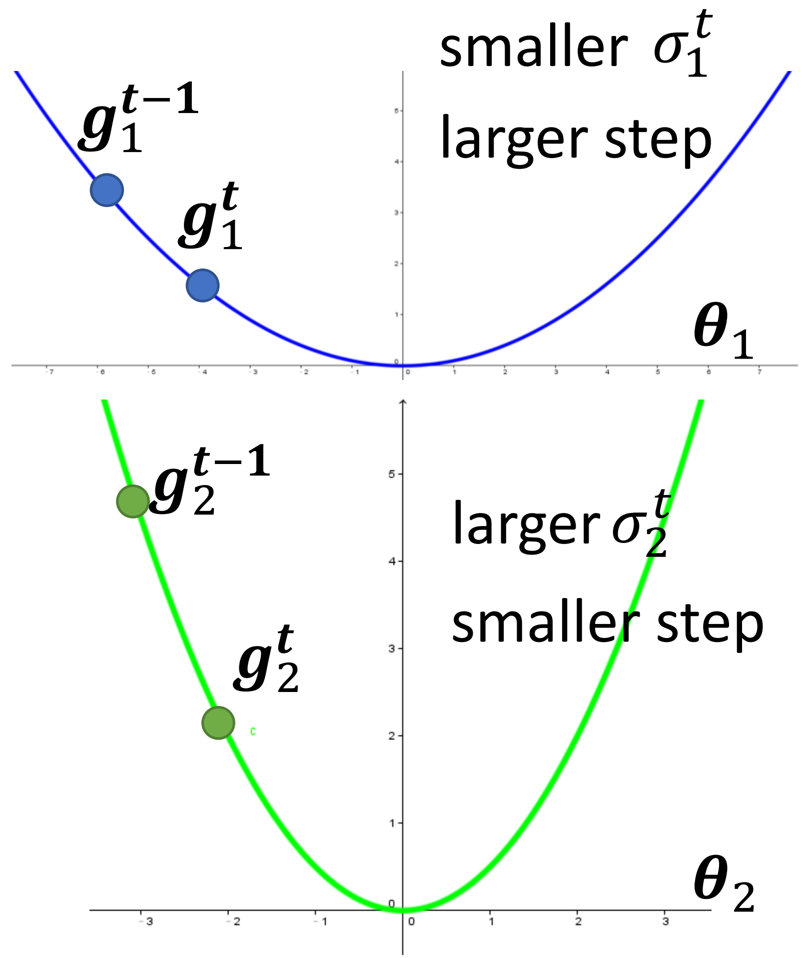
$$\boldsymbol{\theta}_i^{t+1} \leftarrow \boldsymbol{\theta}_i^{t} - \frac{\eta}{\sigma_i^t}\boldsymbol{g}_i^{t} \qquad \sigma_i^t = \sqrt{\frac{1}{t+1}\sum_{i=0}^{t}\left(\boldsymbol{g}_i^{t}\right)^2}$$

Optimizer —— RMS

$$\boldsymbol{\theta}_i^{t+1} \leftarrow \boldsymbol{\theta}_i^t - \boxed{\frac{\eta}{\sigma_i^t}}\boldsymbol{g}_i^t$$

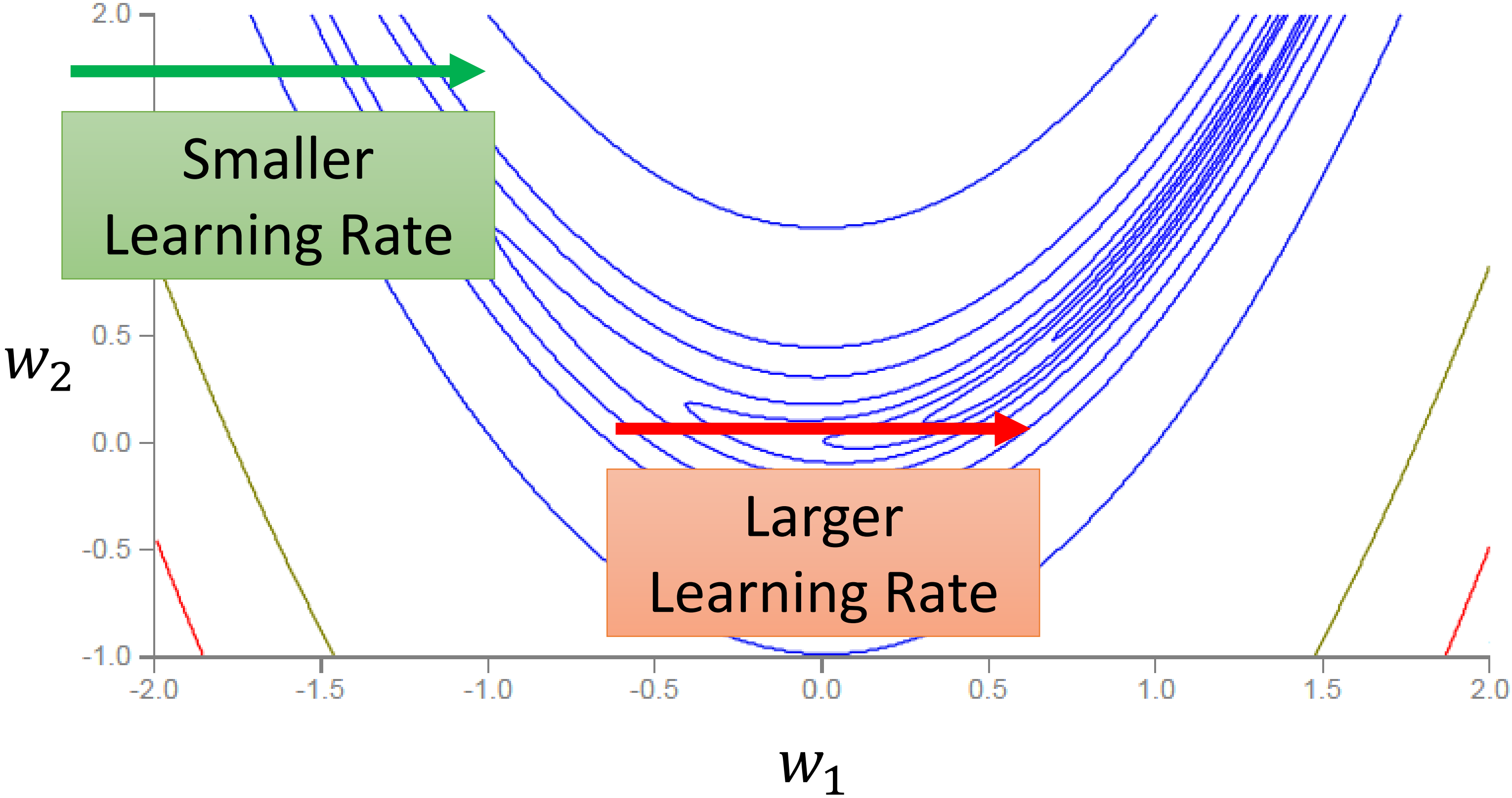
$$\sigma_i^t = \sqrt{\frac{1}{t+1}\sum_{i=0}^{t}\left(\boldsymbol{g}_i^t\right)^2}$$

Used in **Adagrad**



Learning rate adapts dynamically

Error Surface can be very complex.



Optimizer —— RMSProp

$$\boldsymbol{\theta}_i^{t+1} \leftarrow \boldsymbol{\theta}_i^{t} - \boxed{\frac{\eta}{\sigma_i^t}}\boldsymbol{g}_i^t$$

$$\boldsymbol{\theta}_i^{1} \leftarrow \boldsymbol{\theta}_i^{0} - \frac{\eta}{\sigma_i^0}\boldsymbol{g}_i^{\mathbf{0}} \qquad \sigma_i^0 = \sqrt{\left(\boldsymbol{g}_i^{\mathbf{0}}\right)^2}$$

$$0 < \alpha < 1$$

$$\boldsymbol{\theta}_i^{2} \leftarrow \boldsymbol{\theta}_i^{1} - \frac{\eta}{\sigma_i^1}\boldsymbol{g}_i^{\mathbf{1}} \qquad \sigma_i^1 = \sqrt{\alpha\left(\sigma_i^0\right)^2 + (1-\alpha)\left(\boldsymbol{g}_i^{\mathbf{1}}\right)^2}$$

$$\boldsymbol{\theta}_i^{3} \leftarrow \boldsymbol{\theta}_i^{2} - \frac{\eta}{\sigma_i^2}\boldsymbol{g}_i^{\mathbf{2}} \qquad \sigma_i^2 = \sqrt{\alpha\left(\sigma_i^1\right)^2 + (1-\alpha)\left(\boldsymbol{g}_i^{\mathbf{2}}\right)^2}$$

$$\vdots$$

$$\boldsymbol{\theta}_i^{t+1} \leftarrow \boldsymbol{\theta}_i^{t} - \frac{\eta}{\sigma_i^t}\boldsymbol{g}_i^{t} \qquad \sigma_i^t = \sqrt{\alpha\left(\sigma_i^{t-1}\right)^2 + (1-\alpha)\left(\boldsymbol{g}_i^{t}\right)^2}$$

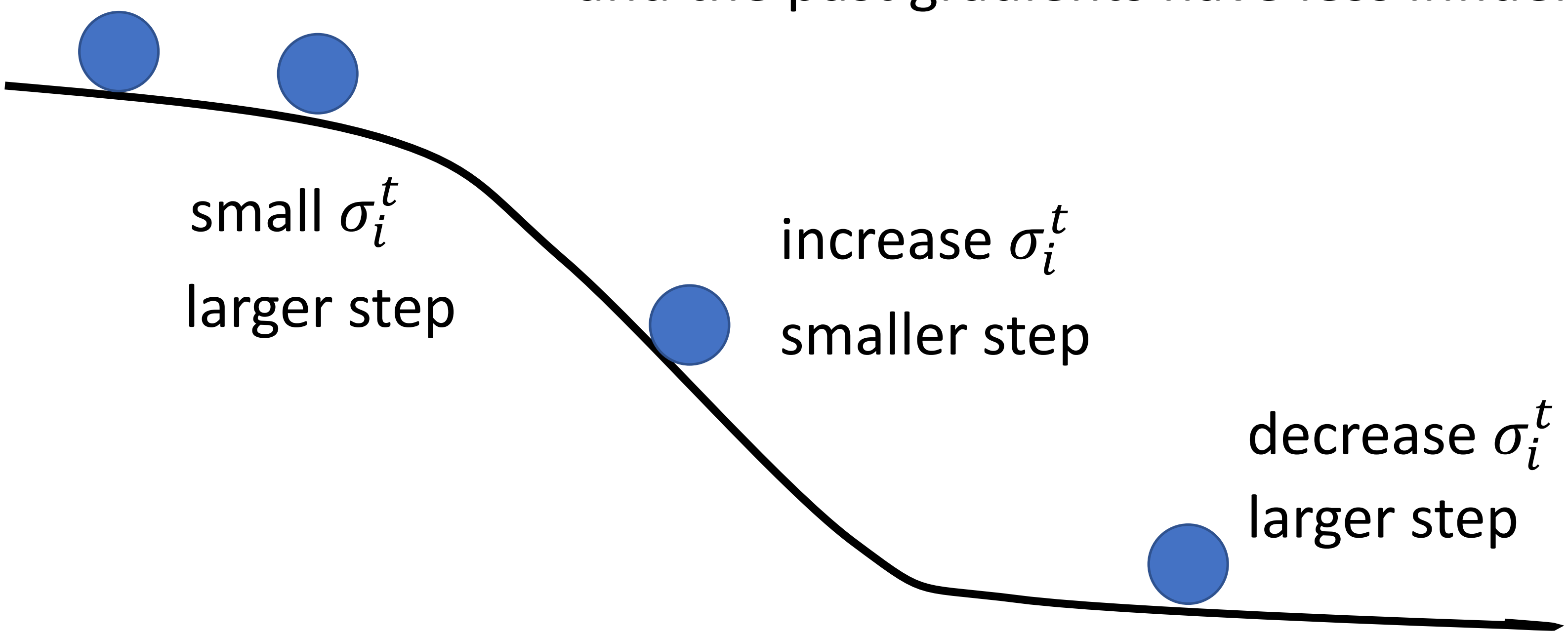
Optimizer —— RMSProp

$$\boldsymbol{g}_i^{1} \; \boldsymbol{g}_i^{2} \; \cdots\cdots \; \boldsymbol{g}_i^{t-1}$$

$$0 < \alpha < 1$$

$$\boldsymbol{\theta}_i^{t+1} \leftarrow \boldsymbol{\theta}_i^{t} - \boxed{\frac{\eta}{\sigma_i^t}} \boldsymbol{g}_i^{t} \qquad \sigma_i^t = \sqrt{\alpha\left(\sigma_i^{t-1}\right)^2 + (1-\alpha)\left(\boldsymbol{g}_i^{t}\right)^2}$$

The recent gradient has larger influence, and the past gradients have less influence.



Optimizer —— Adam

Original paper: https://arxiv.org/pdf/1412.6980.pdf

Adam: RMSProp + Momentum

Algorithm 1: *Adam*, our proposed algorithm for stochastic optimization. See section 2 for details, and for a slightly more efficient (but less clear) order of computation. g_t^2 indicates the elementwise square $g_t \odot g_t$. Good default settings for the tested machine learning problems are $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. All operations on vectors are element-wise. With β_1^t and β_2^t we denote β_1 and β_2 to the power t.

Require: α: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters θ
Require: θ_0: Initial parameter vector
  $m_0 \leftarrow 0$ (Initialize 1st moment vector) → for momentum
  $v_0 \leftarrow 0$ (Initialize 2nd moment vector) → for RMSprop
  $t \leftarrow 0$ (Initialize timestep)
  while θ_t not converged **do**
    $t \leftarrow t + 1$
    $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep t)
    $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
    $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
    $\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
    $\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
    $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
  end while
  return θ_t (Resulting parameters)

Optimizer —— Summary

(Vanilla) Gradient Descent

$$\boldsymbol{\theta}_i^{t+1} \leftarrow \boldsymbol{\theta}_i^{t} - \eta \boldsymbol{g}_i^{t}$$

Various Improvements

$$\boldsymbol{\theta}_i^{t+1} \leftarrow \boldsymbol{\theta}_i^{t} - \frac{\eta^t}{\sigma_i^t} \boldsymbol{m}_i^{t}$$

Learning rate scheduling

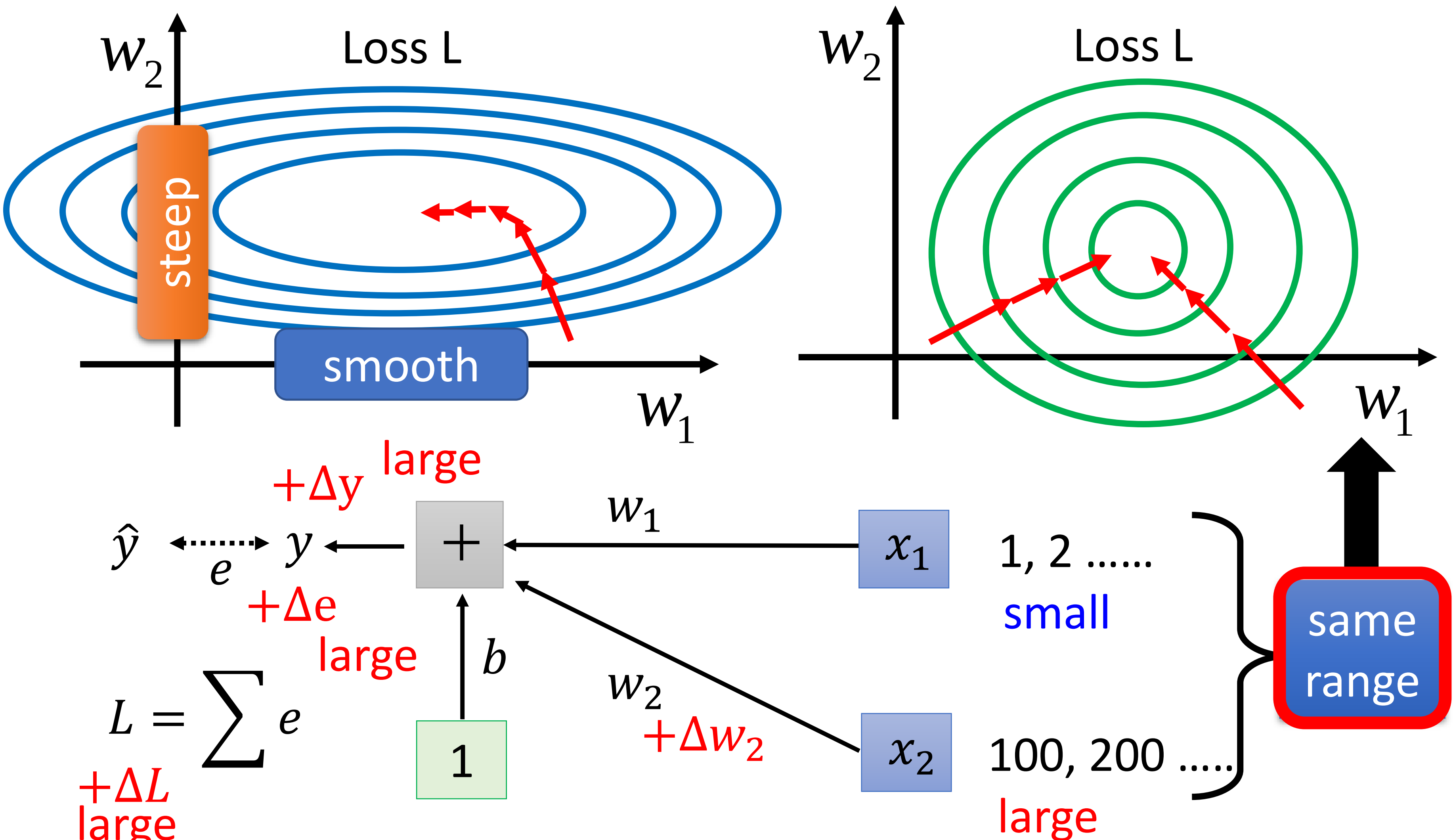
Momentum: weighted sum of the previous gradients

Consider direction

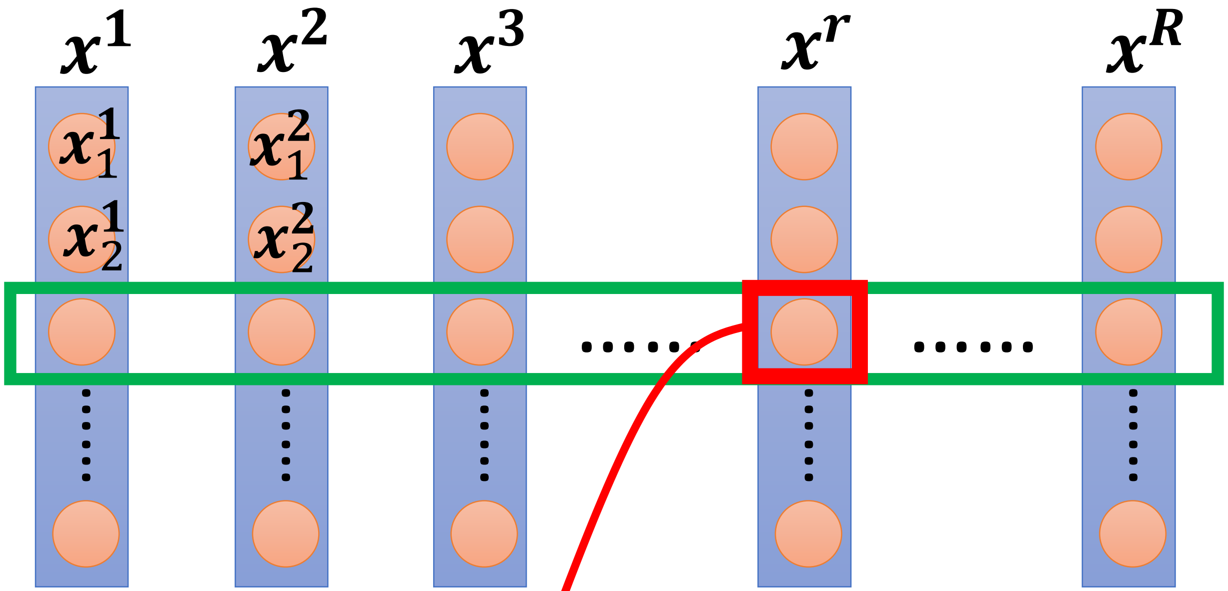
root mean square of the gradients

only magnitude

Changing Landscape



Feature Normalization



For each dimension i:

mean: m_i

standard deviation: σ_i

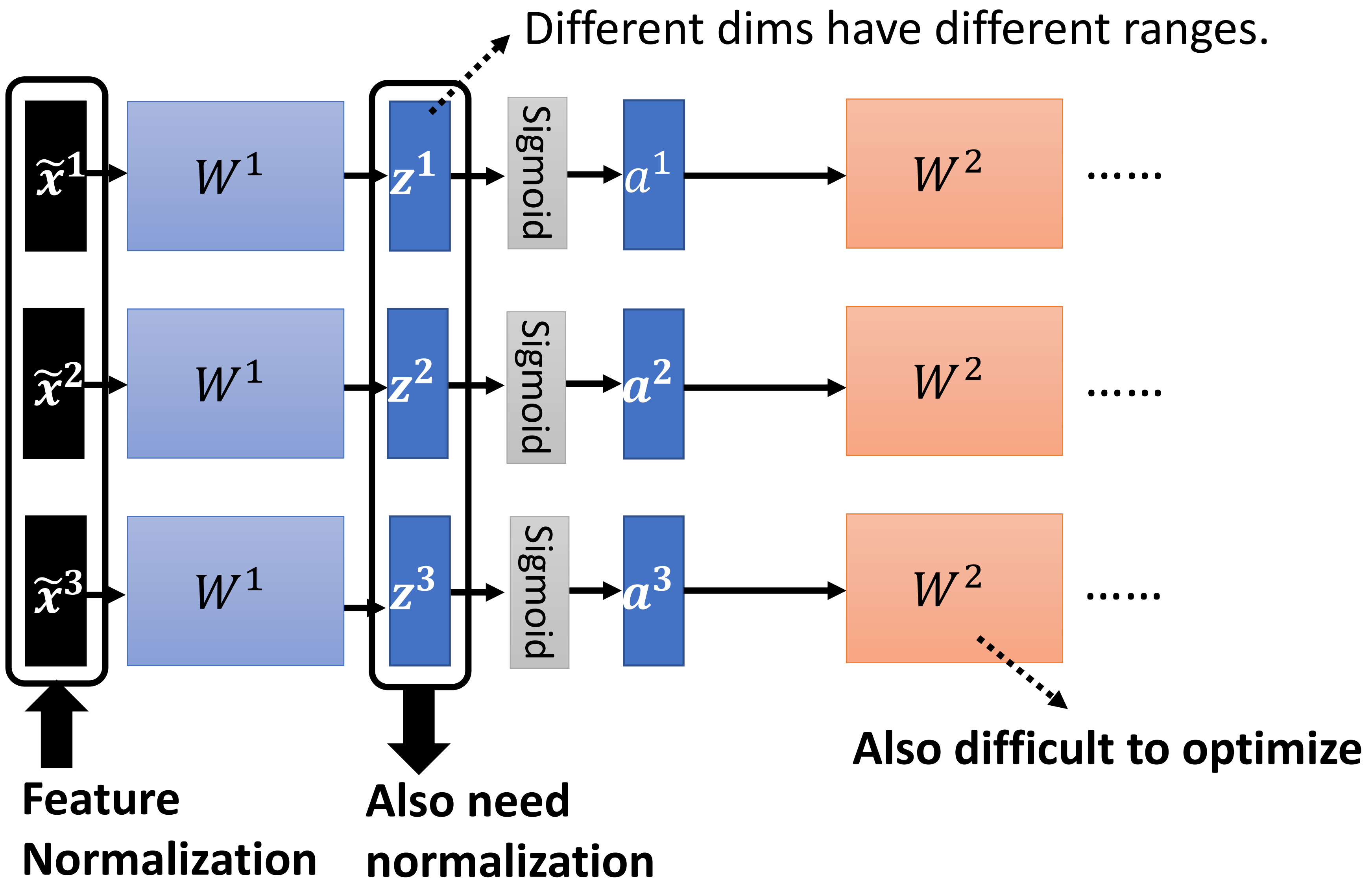
$$\tilde{x}_i^r \leftarrow \frac{x_i^r - m_i}{\sigma_i}$$

The means of all dims are 0, and the variances are all 1

In general, feature normalization makes gradient descent converge faster.

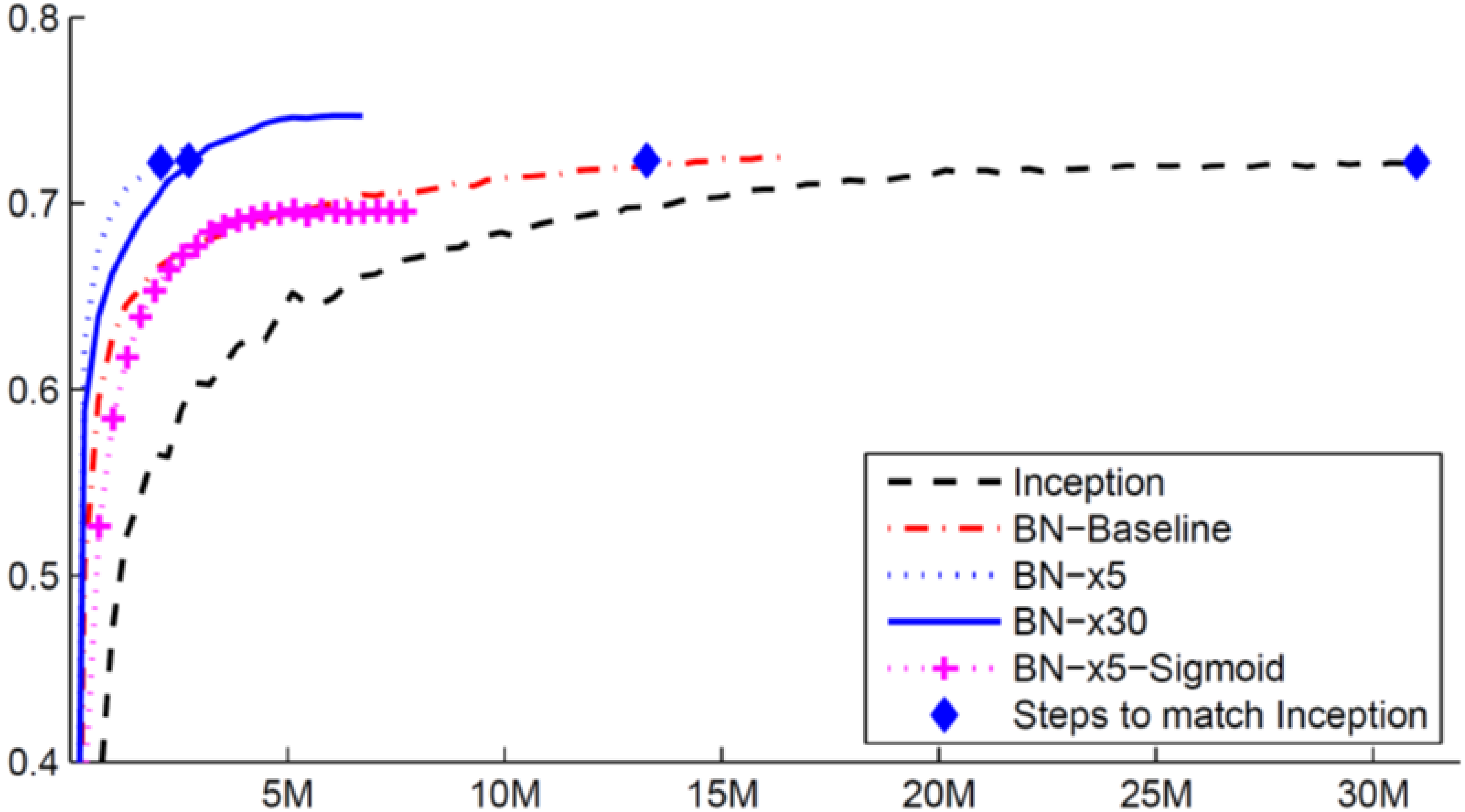
Normalization

Considering Deep Learning



Original paper: https://arxiv.org/abs/1502.03167

Batch normalization



Experimental results (and theoretically analysis) support batch normalization change the landscape of error surface.

Thanks