Deep Learning Crash Course

DLCC

Hui Xue

Fall 2022

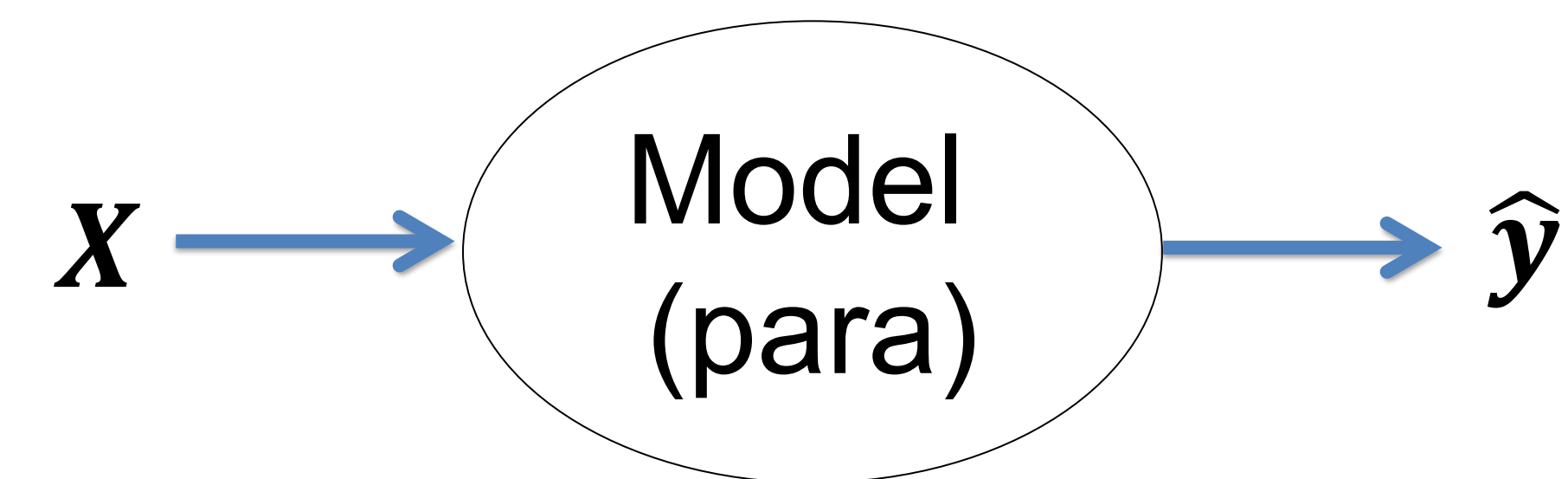# Outline

- Loss function for classification
- Gradient descent
- Bias and variance

Given a model and its parameters, how do we know this model gives correct prediction?

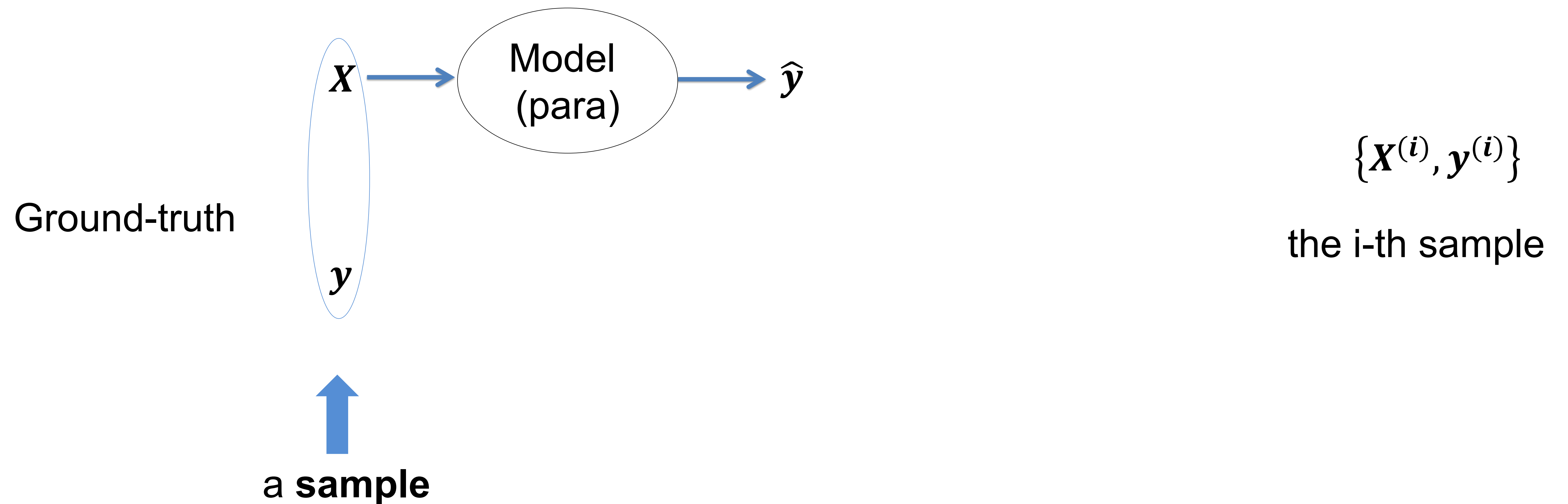$$X \longrightarrow \boxed{\text{Model (para)}} \longrightarrow \hat{y}$$

Given a model and its parameters, how do we know this model gives correct prediction?

$$X \longrightarrow \text{Model (para)} \longrightarrow \widehat{y}$$

Given a model and its parameters, how do we know this model gives correct prediction?

$$\{X^{(i)}, y^{(i)}\}$$

the i-th sample

Given a model and its parameters, how do we know this model gives correct prediction?

Ground-truth

a **sample**

$$\{X^{(i)}, y^{(i)}\}$$

the i-th sample

Given a model and its parameters, how do we know this model gives correct prediction?

$$\{X^{(i)}, y^{(i)}\}$$

the i-th sample

Ground-truth

X → Model (para) → $\hat{y}$

y

Loss func

a **sample**

Given a model and its parameters, how do we know this model gives correct prediction?

$\{X^{(i)}, y^{(i)}\}$

the i-th sample

Cross-entropy

Given two probability distribution p and q, p is the target distribution and q is the model predicted distribution:

$$H(p,q) = - \sum_{for\ all\ possible\ x} p(x)log[q(x;\theta)]$$

- Measure how close two distributions are (not 100% rigorous)
- Minimized if p=q

$$H(p,q) = H(p) + D_{KL}(p \parallel q)$$

$D_{KL}(p \parallel q)$ is the KL divergence of p and q.

$$D_{KL}(p \parallel q) \geq 0$$

$D_{KL}(p \parallel q) = 0$ if and only if p=q

$H(p)$ is the entropy of distribution p

$$H(p,q) \neq H(q,p)$$

$$H(p,q) = -\sum_x p(x) \log[q(x)]$$

$$= -\sum_x p(x) \log\left[\frac{p(x)}{p(x)} q(x)\right]$$

$$= -\sum_x p(x) \left\{ \log[p(x)] + \log\left[\frac{q(x)}{p(x)}\right] \right\}$$

$$= \underbrace{-\sum_x p(x) \log[p(x)]}_{H(p)} \underbrace{- \sum_x p(x) \log\left[\frac{q(x)}{p(x)}\right]}_{D_{KL}(p \parallel q)}$$

$$= -\sum_x p(x) \log[p(x)] + \sum_x p(x) \log\left[\frac{p(x)}{q(x)}\right]$$

Entropy of distribution p

Binary Cross-entropy loss

For the binary classification,

y = 1 for the "Yes" class and 0 for the "No" class

$\hat{y} \in [0, 1]$, probability belonging to the "Yes" class
$1 - \hat{y}$ for the "No" class

Then, the cross entropy (in this case, the BCE loss) is:

$$L_{BCE\_loss}(y, \hat{y}) = -[y \cdot log(\hat{y}) + (1 - y) \cdot log(1 - \hat{y})]$$

To minimize the BCE loss,

if $y = 0$, $\hat{y} = 0$ will reach minimum of log(1)=0

if $y = 1$, $\hat{y} = 1$ will reach minimum of log(1)=0

For all other cases, $L_{BCE\_loss} > 0$

Cross-entropy loss for multi-class

For the multi-class classification,

$$y = \begin{cases} CH3, & 1 \quad 0 \quad 0 \quad 0 \quad 0 \\ CH2, & 0 \quad 1 \quad 0 \quad 0 \quad 0 \\ CH4, & 0 \quad 0 \quad 1 \quad 0 \quad 0 \\ SAX, & 0 \quad 0 \quad 0 \quad 1 \quad 0 \\ Other, & 0 \quad 0 \quad 0 \quad 0 \quad 1 \end{cases}$$

y is one-hot encoding for the correct class

$$y_k = \begin{cases} CH3, 0 \\ CH2, 1 \\ CH4, 2 \\ SAX, 3 \\ Other, 4 \end{cases}$$

y_k is the index of correct class for sample y

$\hat{y}$ is a [K,1] vector for the probability of X belonging to each lass

$\hat{y}[0]$ is a scalar to predict X being CH3
$\hat{y}[1]$ is a scalar to predict X being CH2
$\hat{y}[2]$ is a scalar to predict X being CH4
… …

$\hat{y}[y_k]$ is a scalar to predict X being class y_k

Cross-entropy loss

For the multi-class classification,

$$L_{CE\_loss}(y, \widehat{y}) = -\sum_{k=0}^{4} y[k]\, log(\widehat{y}[k])$$

$\widehat{y}$ is a [K,1] vector for the probability of X being each lass

$\widehat{y}[y_k]$ is a scalar to predict X being class y_k

y is one-hot encoding for the correct class

y_k is the index of correct class for sample y

$$L_{CE\_loss}(y, \widehat{y}) = -log(\widehat{y}[y_k])$$

Only one term left!

To minimize the cross-entropy loss,

We need $\widehat{y}[y_k] = 1$; for all other cases, $L_{CE\_loss} > 0$

Cross-entropy loss

	y		$\hat{y}$
CH2	0		0.2
CH3	0		0.3
CH4	1		0.1
SAX	0		0.15
Other	0		0.25

$$L_{CE\_loss}(y, \hat{y}) = -\sum_{k=0}^{4} y[k] \, log(\hat{y}[k])$$

$$= -[0 \times \log(0.2)$$
$$+ 0 \times \log(0.3)$$
$$+ 1 \times \log(0.1)$$
$$+ 0 \times \log(0.15)$$
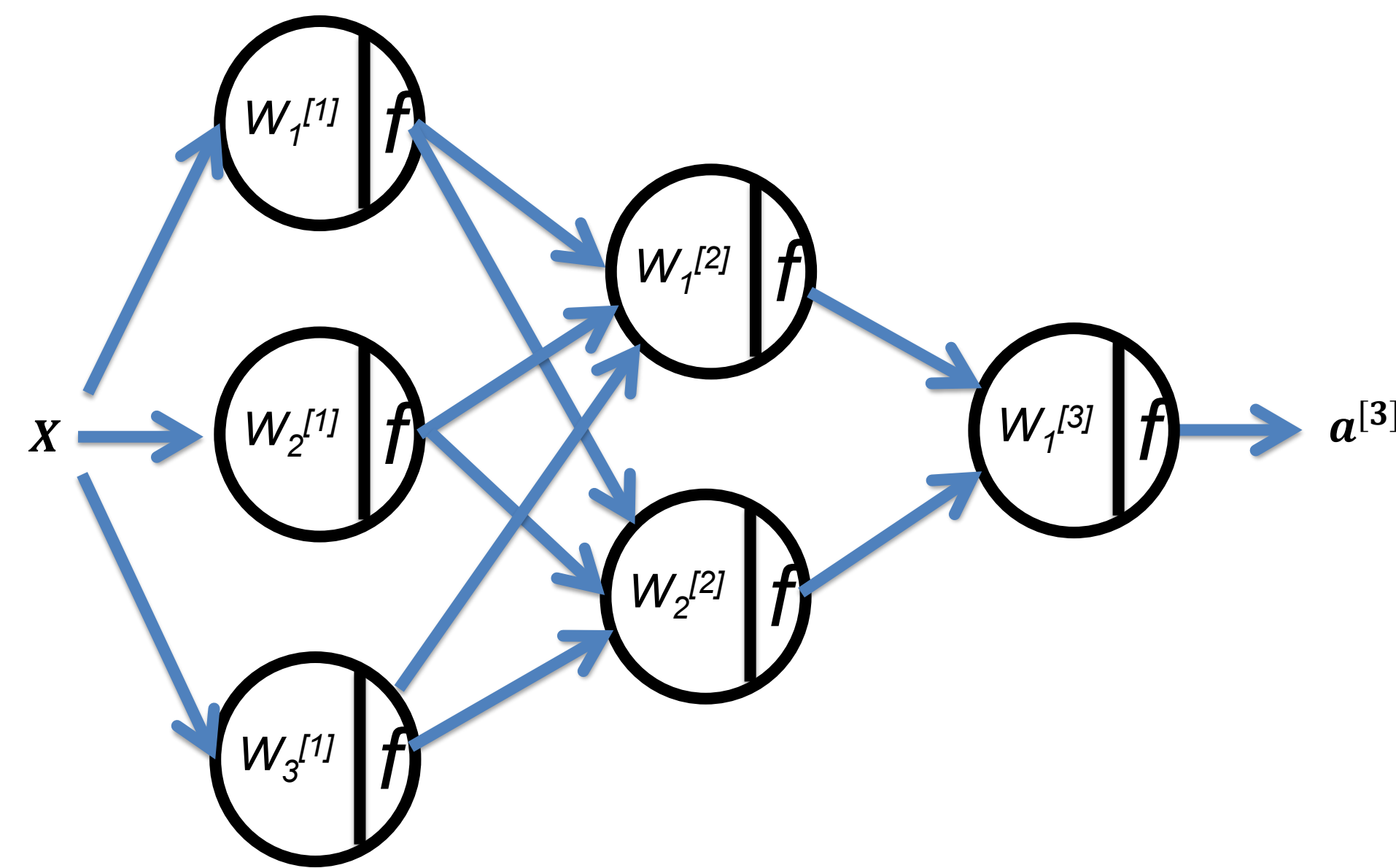$$+ 0 \times \log(0.25)]$$
$$= -\log(0.1) = 2.3$$

Cross-entropy loss

	y		$\widehat{y}$
CH2	0		0.02
CH3	0		0.1
CH4	1		0.8
SAX	0		0.05
Other	0		0.03

$$L_{CE\_loss}(y, \widehat{y}) = -\sum_{k=0}^{4} y[k] \, log(\widehat{y}[k])$$

$$= -[0 \times \log(0.02)$$
$$+ 0 \times \log(0.1)$$
$$+ 1 \times \log(0.8)$$
$$+ 0 \times \log(0.05)$$
$$+ 0 \times \log(0.03)]$$
$$= -\log(0.8) = 0.22$$

$$Z^{[1]} = W^{[1]}X + b^{[1]}$$

$$a^{[1]} = f(Z^{[1]})$$

$$Z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$$

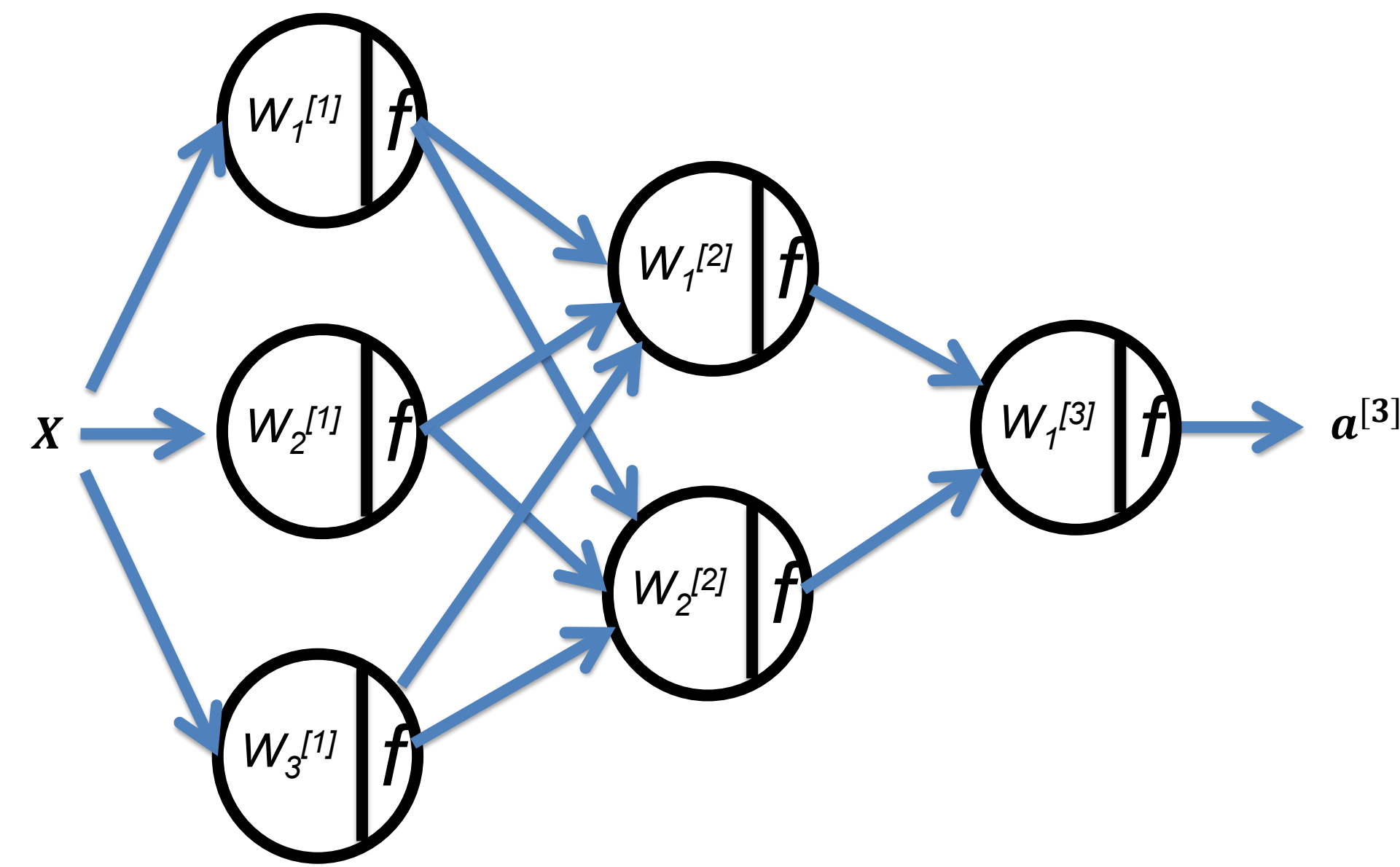$$a^{[2]} = f(Z^{[2]})$$

$$Z^{[3]} = W^{[3]}a^{[2]} + b^{[3]}$$

$$\hat{y} = a^{[3]} = f(Z^{[3]})$$

$$L_{BCE\_loss}(y, \hat{y})$$

$$L_{CE\_loss}(y, \hat{y})$$

Model : map input to output

Loss : measure how good the output is, compared to the label

$$W^{[1]}, b^{[1]}$$

$$W^{[2]}, b^{[2]}$$

$$W^{[3]}, b^{[3]}$$

Model : map input to output

$$W^{[1]}, b^{[1]}$$
$$W^{[2]}, b^{[2]}$$
$$W^{[3]}, b^{[3]}$$

Model : map input to output

$$W^{[1]}, b^{[1]}$$

$$W^{[2]}, b^{[2]}$$

$$W^{[3]}, b^{[3]}$$

Model : map input to output

Model : map input to output

$$W^{[1]}, b^{[1]}$$
$$W^{[2]}, b^{[2]}$$
$$W^{[3]}, b^{[3]}$$

X

y

Model (para)

$\widehat{y}$

Optimization

Loss func

$L(y, \widehat{y})$

Backprop:
Compute gradient

Model : map input to output

$$W^{[1]}, b^{[1]}$$
$$W^{[2]}, b^{[2]}$$
$$W^{[3]}, b^{[3]}$$

Update model paras

Optimization

X → Model (para) → $\hat{y}$

y → Loss func → $L(y, \hat{y})$

Backprop: Compute gradient

Gradient descent

J(w)

Initial weight
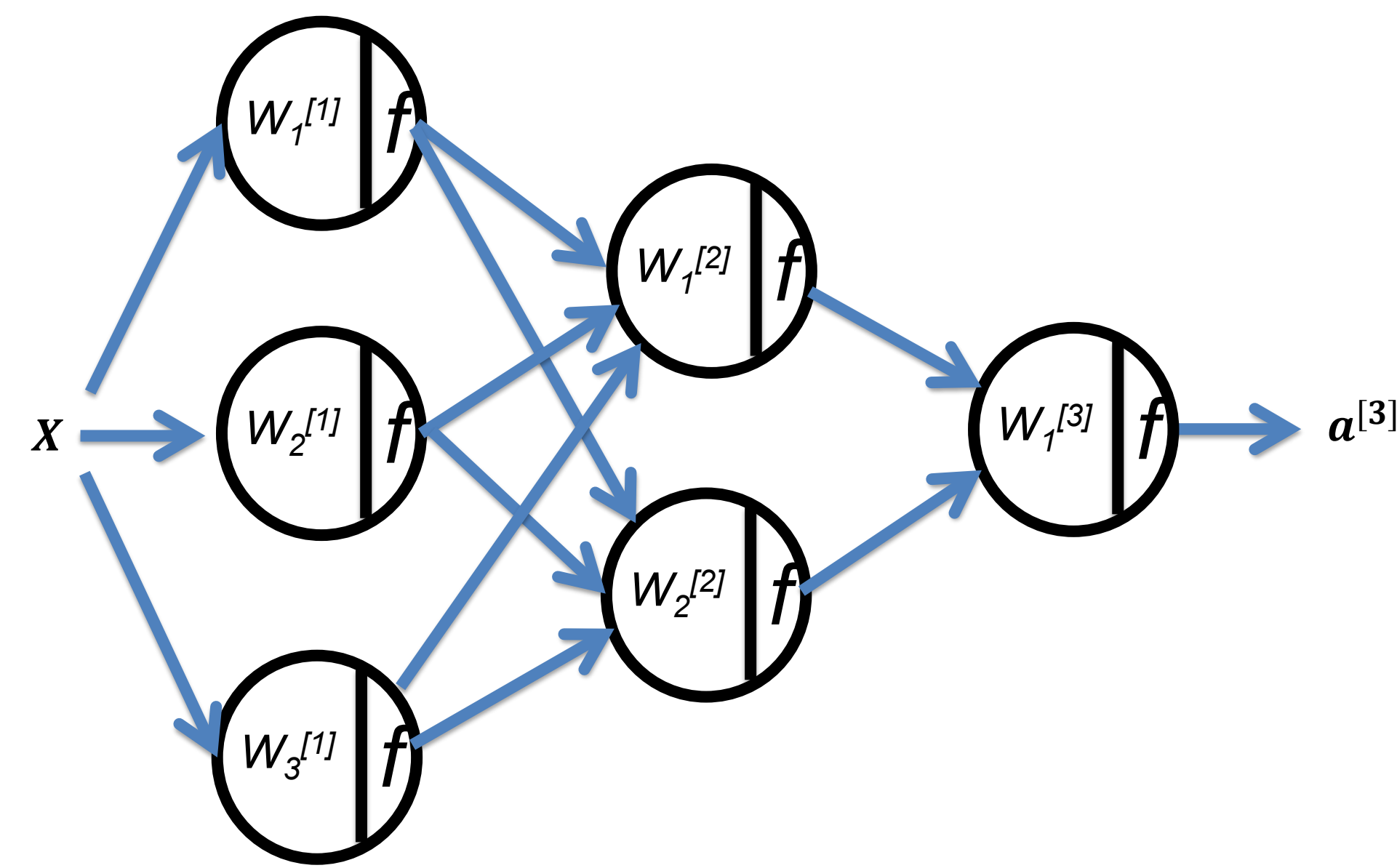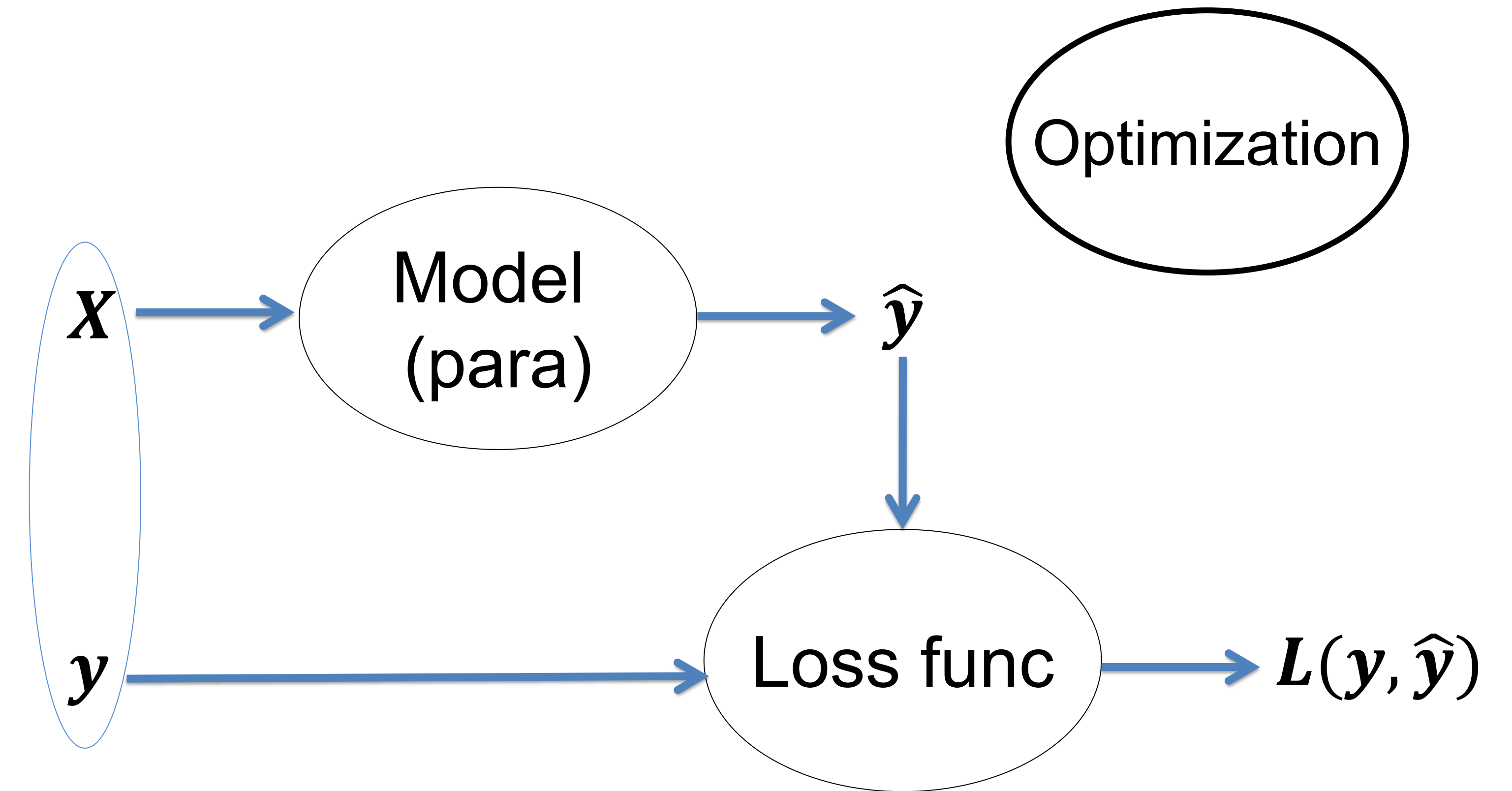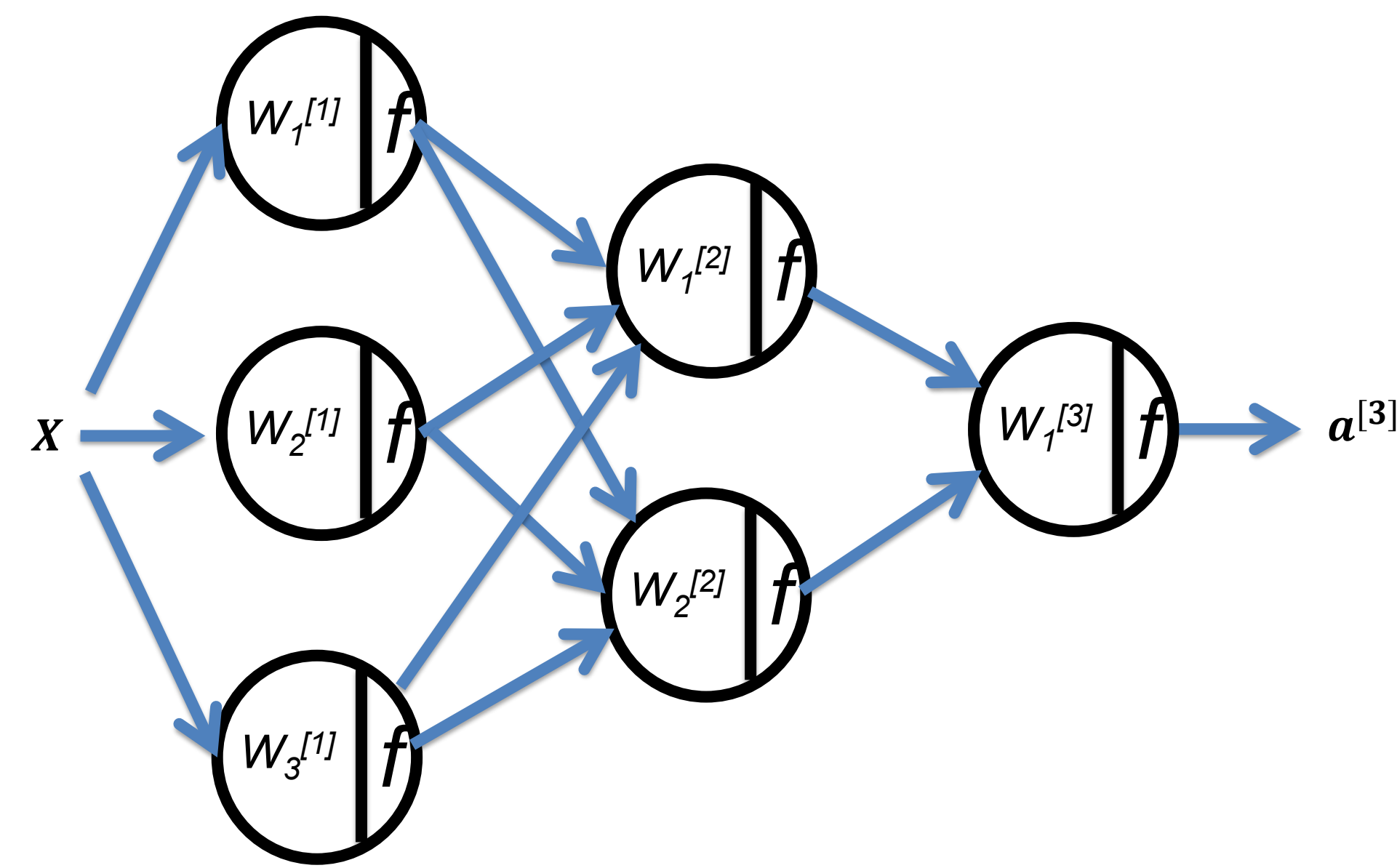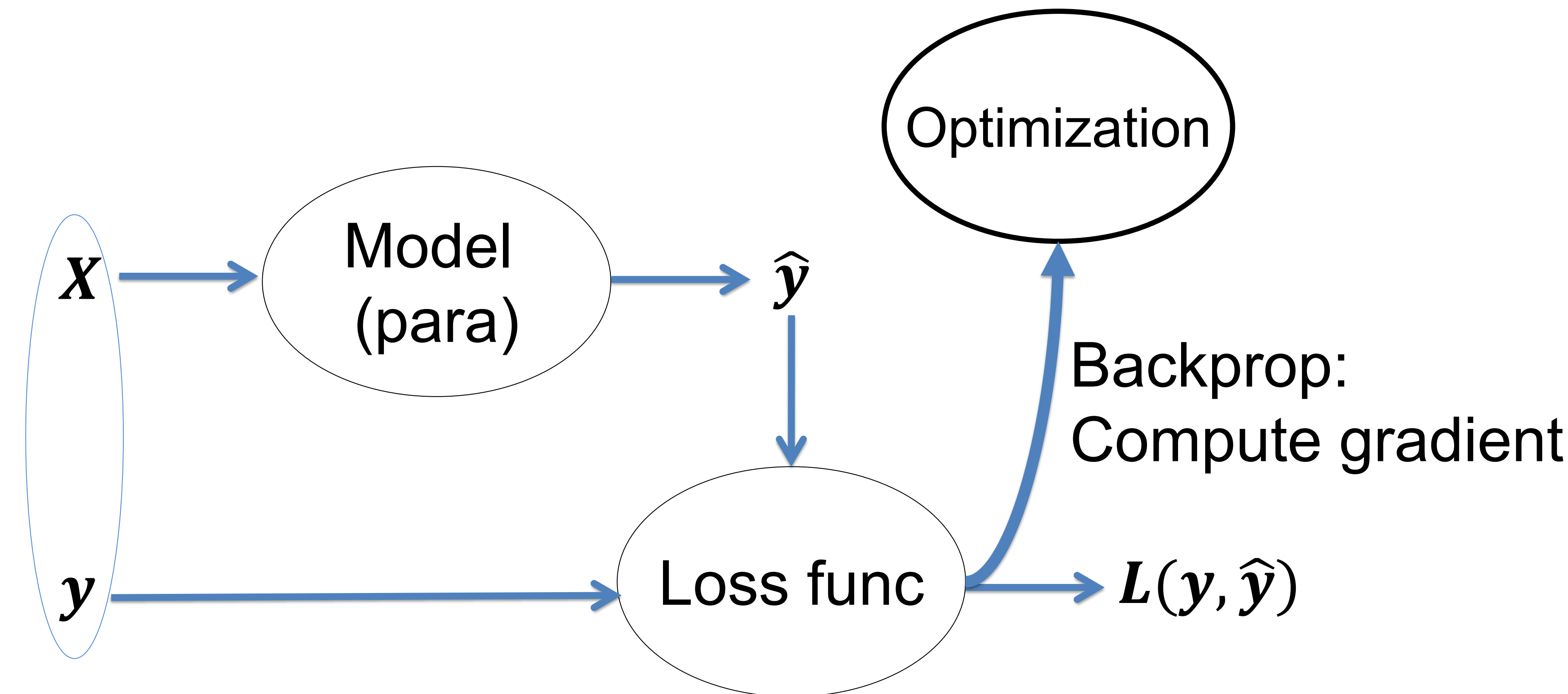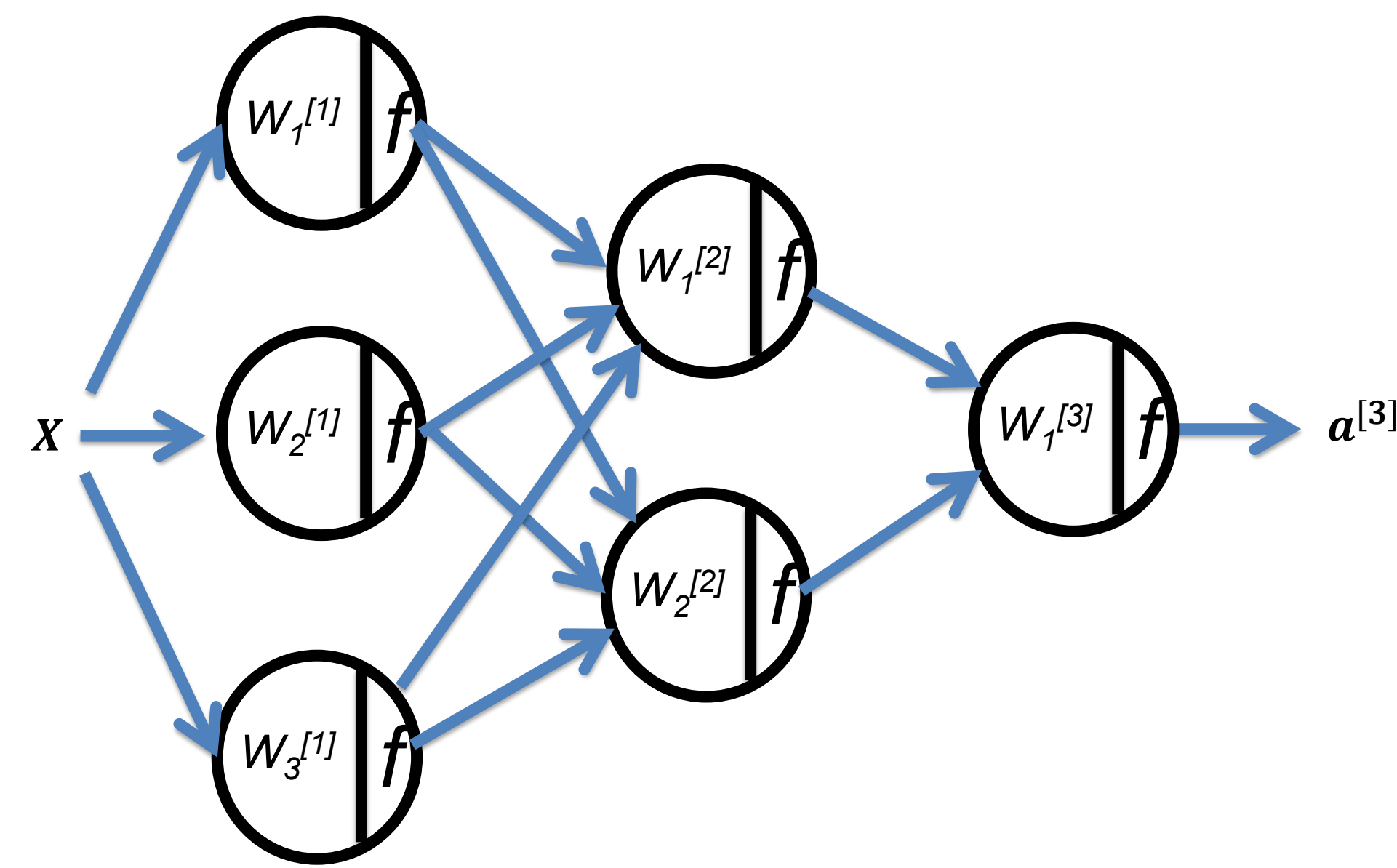
Gradient

To fine minimal point of a function

- Start from an initial point
- Follow the negative gradient direction

Global cost minimum
$J_{min}(w)$

w

Gradient of a function

Gradient of a function indicates the steepest direction to **increase** this function in a neighborhood.
Negative gradient of a function indicates the steepest direction to **decrease** this function in a neighborhood.

f is a scalar function.

In 1-dimension, the derivative of a function:

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

$$\nabla f(\boldsymbol{p}) = \begin{bmatrix} \dfrac{\partial f}{\partial x_0} \\ \dfrac{\partial f}{\partial x_1} \\ ... \\ \dfrac{\partial f}{\partial x_{N-1}} \end{bmatrix}$$

- $\boldsymbol{p} = [\boldsymbol{x_0}, \boldsymbol{x_1}, \boldsymbol{x_2}, ..., \boldsymbol{x_{N-1}}]^{\boldsymbol{T}}$
  is a point in N-dimensional space

- $\nabla f(\boldsymbol{p})$ is the gradient at point $\boldsymbol{p}$.
- Its direction points to the steepest slope to increase the function.
- The steepness of the slope at that point is given by the $|\nabla f(\boldsymbol{p})|$
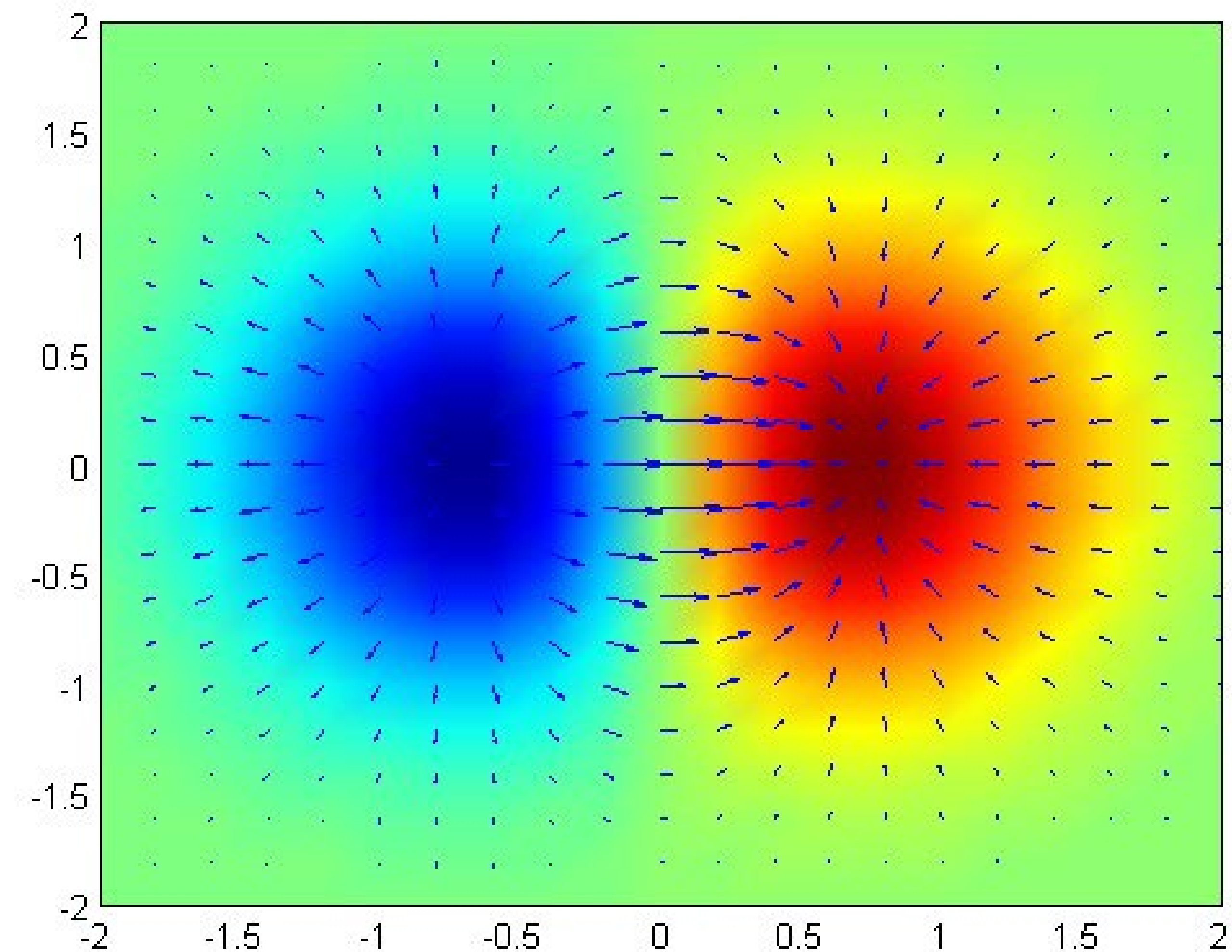
https://en.wikipedia.org/wiki/Gradient#/media/File:Gradient_of_a_Function.tif

13

Gradient descent

Gradient direction
is
the steepest direction to **increase** this function

We want to minimize the loss $L(\boldsymbol{y}, \widehat{\boldsymbol{y}}(\boldsymbol{W}, \boldsymbol{b}))$ by adjusting the model parameters

local minima

For every layer l :

$$\boldsymbol{W}^{[l]} = \boldsymbol{W}^{[l]} - \alpha \frac{\partial L(\boldsymbol{y}, \widehat{\boldsymbol{y}}(\boldsymbol{W}, \boldsymbol{b}))}{\partial \boldsymbol{W}^{[l]}}$$

$$\boldsymbol{b}^{[l]} = \boldsymbol{b}^{[l]} - \alpha \frac{\partial L(\boldsymbol{y}, \widehat{\boldsymbol{y}}(\boldsymbol{W}, \boldsymbol{b}))}{\partial \boldsymbol{b}^{[l]}}$$

α : **learning rate**

Guarantee to find a local minima
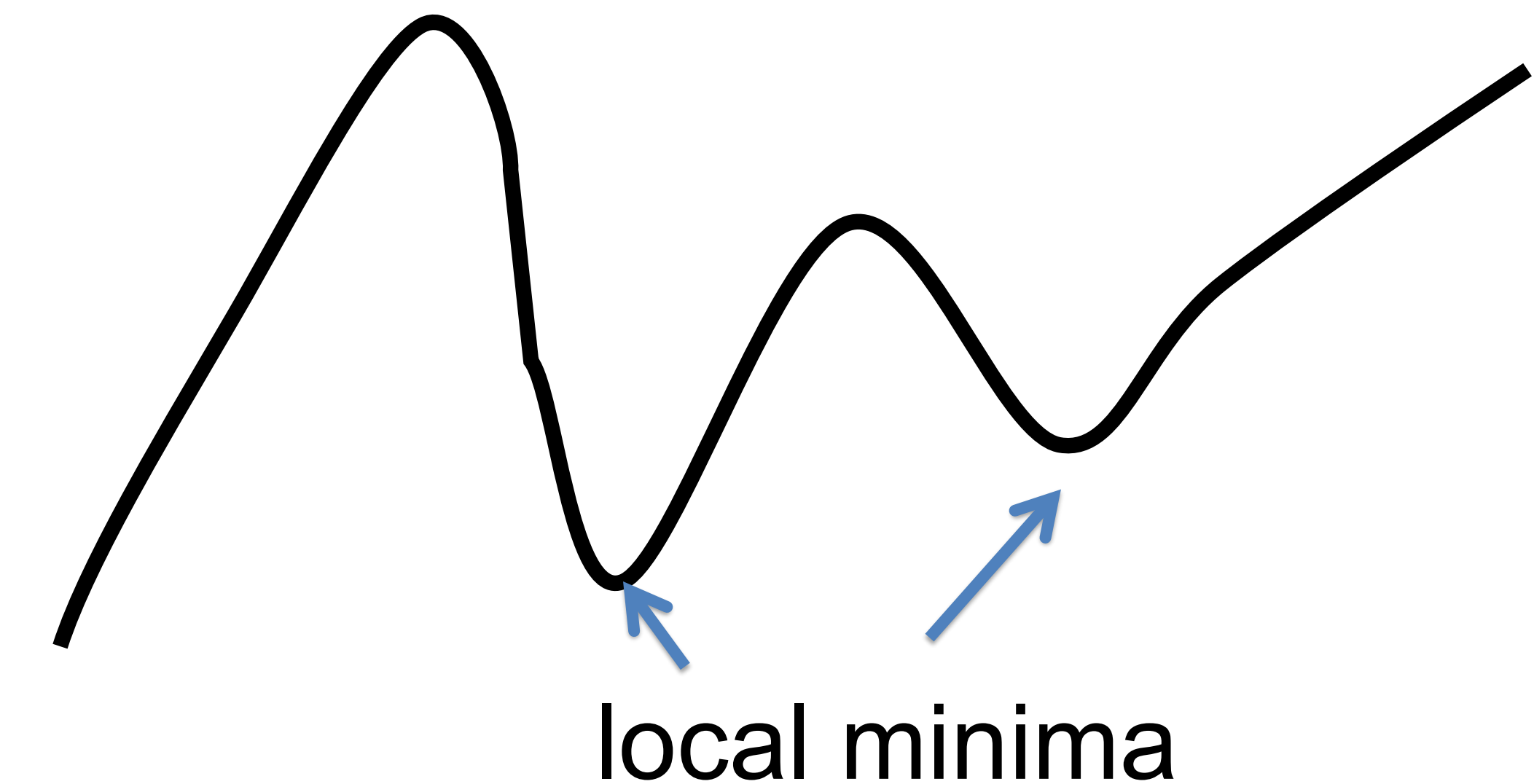
Gradient Descent (GD) over a set of samples

We have M samples

Empirical loss on measured data $= \frac{1}{M}\sum_{i=0}^{M-1} L\left(\boldsymbol{y}^{(i)}, \widehat{\boldsymbol{y}}^{(i)}(\boldsymbol{W}, \boldsymbol{b})\right)$

$$\frac{\partial L(\boldsymbol{y}, \widehat{\boldsymbol{y}}(\boldsymbol{W}, \boldsymbol{b}))}{\partial \boldsymbol{W}^{[l]}} = \frac{1}{M}\sum_{i=0}^{M-1} \frac{\partial L\left(\boldsymbol{y}^{(i)}, \widehat{\boldsymbol{y}}^{(i)}(\boldsymbol{W}, \boldsymbol{b})\right)}{\partial \boldsymbol{W}^{[l]}}$$

Find best parameters to minimize the mean loss across all training samples

$$\frac{\partial L(\boldsymbol{y}, \widehat{\boldsymbol{y}}(\boldsymbol{W}, \boldsymbol{b}))}{\partial \boldsymbol{b}^{[l]}} = \frac{1}{M}\sum_{i=0}^{M-1} \frac{\partial L\left(\boldsymbol{y}^{(i)}, \widehat{\boldsymbol{y}}^{(i)}(\boldsymbol{W}, \boldsymbol{b})\right)}{\partial \boldsymbol{b}^{[l]}}$$

Gradient descent

Initialize weights and bias

for iter in range(t):

Evaluate loss function (forward pass) over all samples
$$L = \frac{1}{M} \sum_{i=0}^{M-1} L(\boldsymbol{y}^{(i)}, \widehat{\boldsymbol{y}}^{(i)}(\boldsymbol{W}, \boldsymbol{b}))$$

Compute gradient
$$\frac{\partial L}{\partial \boldsymbol{W}^{[l]}}, \frac{\partial L}{\partial \boldsymbol{b}^{[l]}}$$

Update parameter

$$\boldsymbol{W}^{[l]} = \boldsymbol{W}^{[l]} - \alpha \frac{\partial L}{\partial \boldsymbol{W}^{[l]}}$$
$$\boldsymbol{b}^{[l]} = \boldsymbol{b}^{[l]} - \alpha \frac{\partial L}{\partial \boldsymbol{b}^{[l]}}$$

Stochastic Gradient Descent (SGD)

GD performs one parameter update step after going through entire training set

→Slow if M is large
→Gradient update may lack "exploration"

Initialize weights and bias
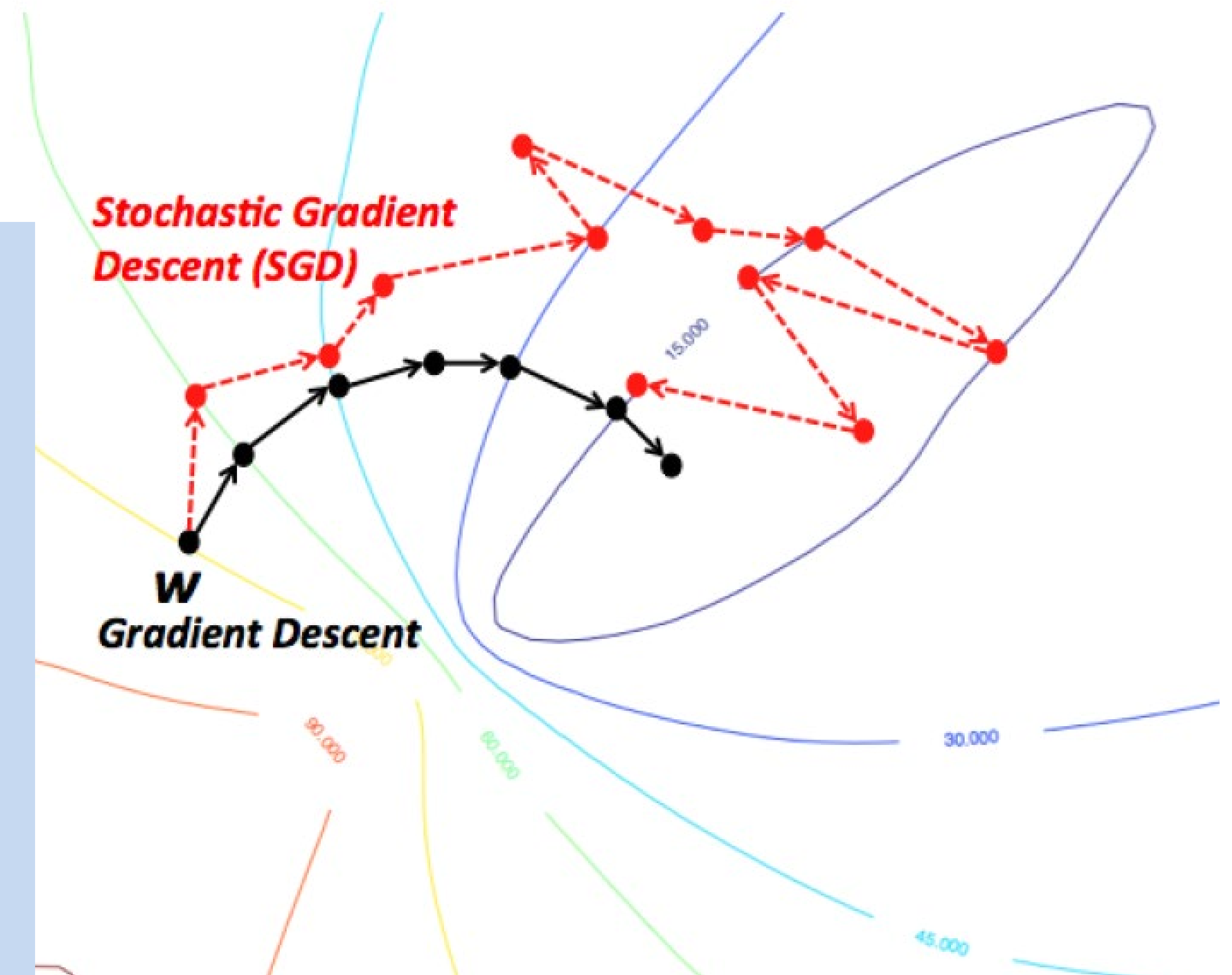
Random shuffle dataset

for epoch in range(E):

    select one sample with index i

    Evaluate loss function (forward pass) at this sample
       $L = L(\boldsymbol{y}^{(i)}, \widehat{\boldsymbol{y}}^{(i)}(\boldsymbol{W}, \boldsymbol{b}))$

    Compute gradient $\frac{\partial L}{\partial \boldsymbol{W}^{[l]}}, \frac{\partial L}{\partial \boldsymbol{b}^{[l]}}$

    Update parameter $\boldsymbol{W}^{[l]} = \boldsymbol{W}^{[l]} - \alpha \frac{\partial L}{\partial \boldsymbol{W}^{[l]}}, \boldsymbol{b}^{[l]} = \boldsymbol{b}^{[l]} - \alpha \frac{\partial L}{\partial \boldsymbol{b}^{[l]}}$

Stochastic Gradient Descent (SGD)

W
Gradient Descent

15,000
30,000
45,000
90,000
60,000

Mini-Batch SGD

SGD can have too much "noise" during convergence
Not fully utilize the computing hardware

Initialize weights and bias

Random shuffle dataset

BatchSize (B) = 32

for epoch in range(E):

    select #BatchSize **samples**

    Evaluate loss function (forward pass) at this ~~sample~~ **batch**
$$L = \frac{1}{B} \sum_{i=0}^{B-1} L(\boldsymbol{y}^{(i)}, \widehat{\boldsymbol{y}}^{(i)}(\boldsymbol{W}, \boldsymbol{b}))$$

    Compute gradient $\frac{\partial L}{\partial \boldsymbol{W}^{[l]}}, \frac{\partial L}{\partial \boldsymbol{b}^{[l]}}$

    Update parameter $\boldsymbol{W}^{[l]} = \boldsymbol{W}^{[l]} - \alpha \frac{\partial L}{\partial \boldsymbol{W}^{[l]}}, \ \boldsymbol{b}^{[l]} = \boldsymbol{b}^{[l]} - \alpha \frac{\partial L}{\partial \boldsymbol{b}^{[l]}}$

- Batch size often is limited by the GPU RAM

- Different way to select a batch, e.g. sequential, random, fixed step size etc.

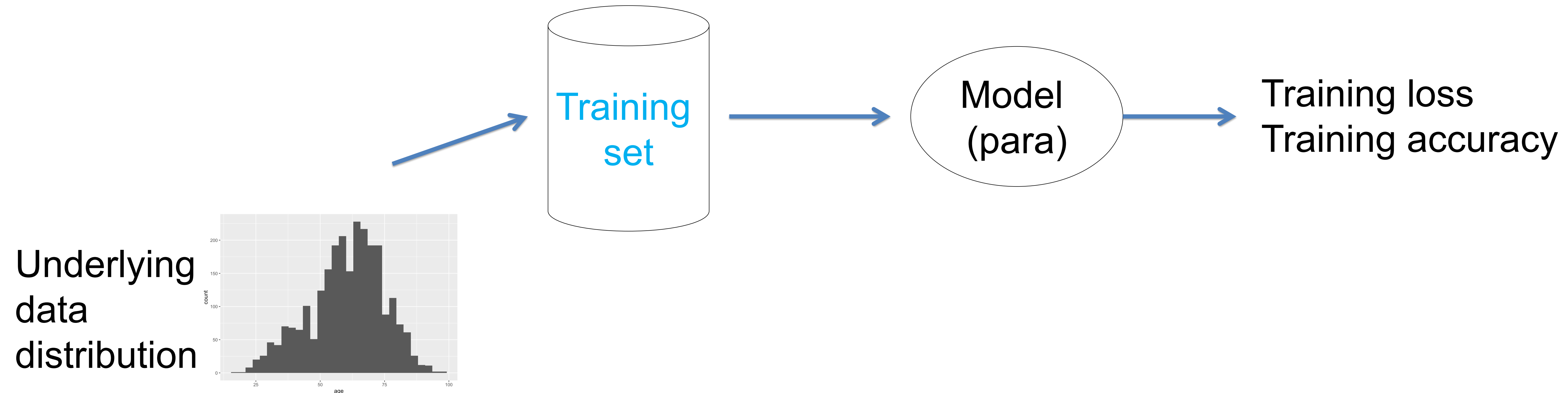- Every iteration can have a different B

18

Larger batch size, higher learning rate
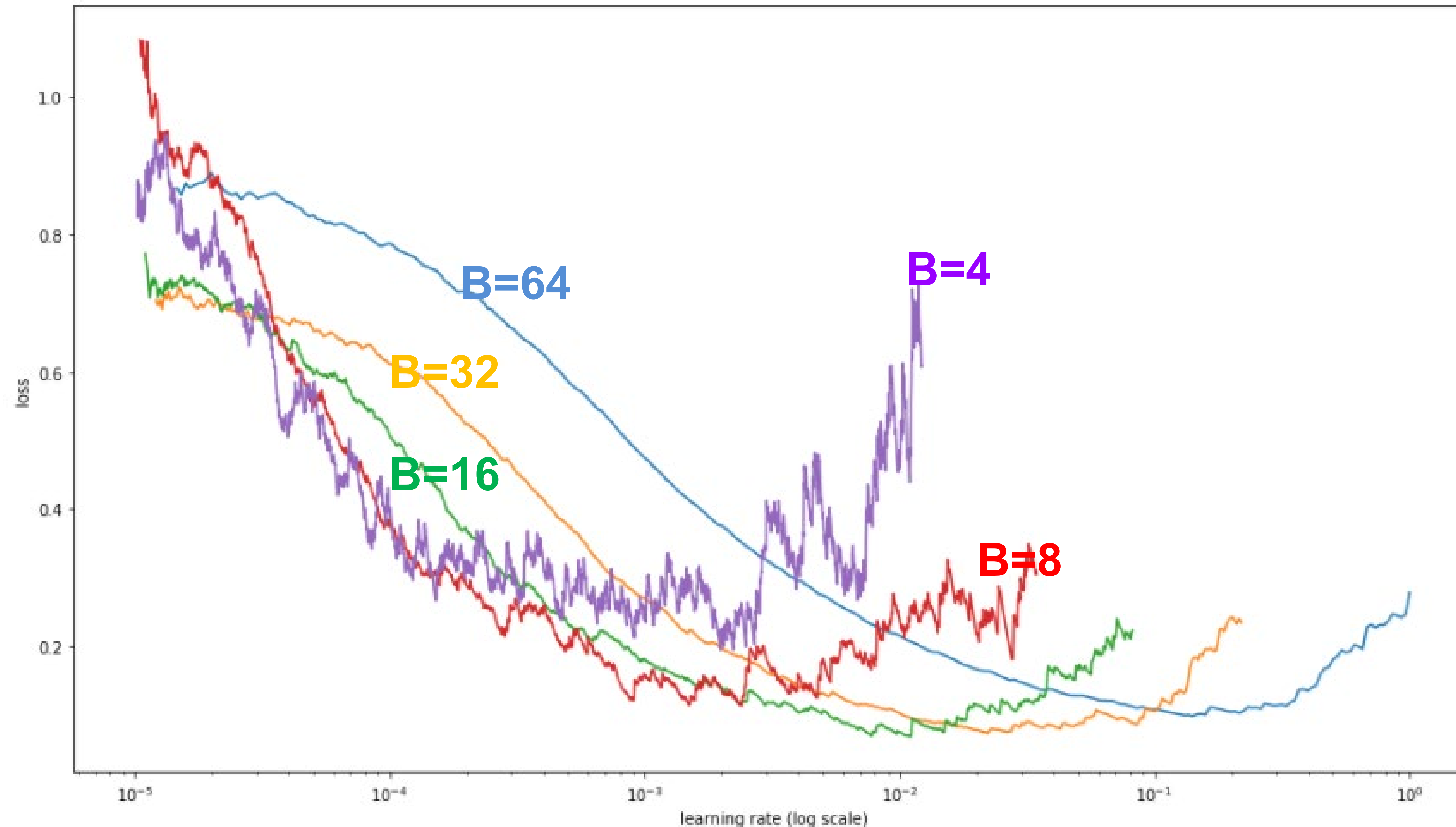
- Larger BatchSize, better estimation of gradient

- Larger BatchSize, less exploration

- High learning rate for small BatchSize can lead to failed convergence

- Overall, set BatchSize large, subject to the GPU RAM limit

- More on how to find good learning rate ...

https://miguel-data-sc.github.io/2017-11-05-first/#:~:text=For%20the%20ones%20unaware%2C%20general,descent%20(batch%20size%201).   https://arxiv.org/abs/1506.01186

Model Generalization

Underlying
data
distribution

We don't have this
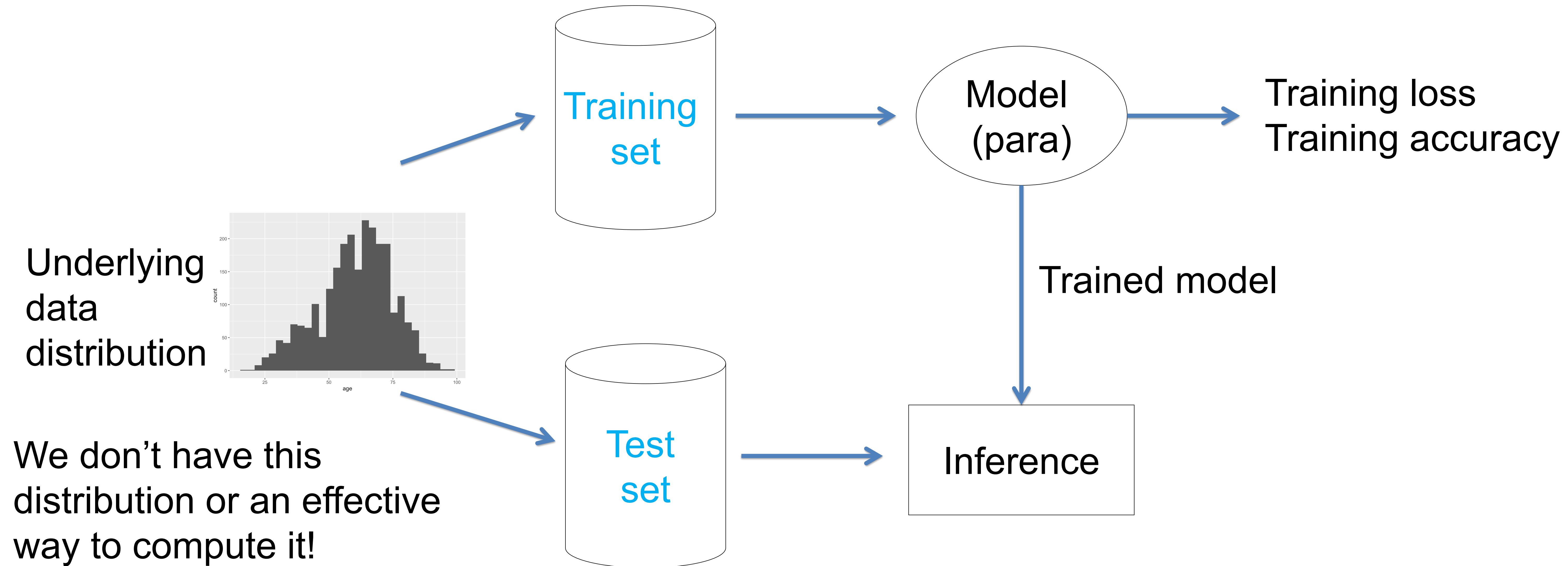distribution or an effective
way to compute it!

- Training and test sets are sampled from the same distribution, i.i.d. (independent and identically distributed)
- We care test/generalization performance; that is, the performance of training model on a new dataset

Model Generalization

Underlying data distribution

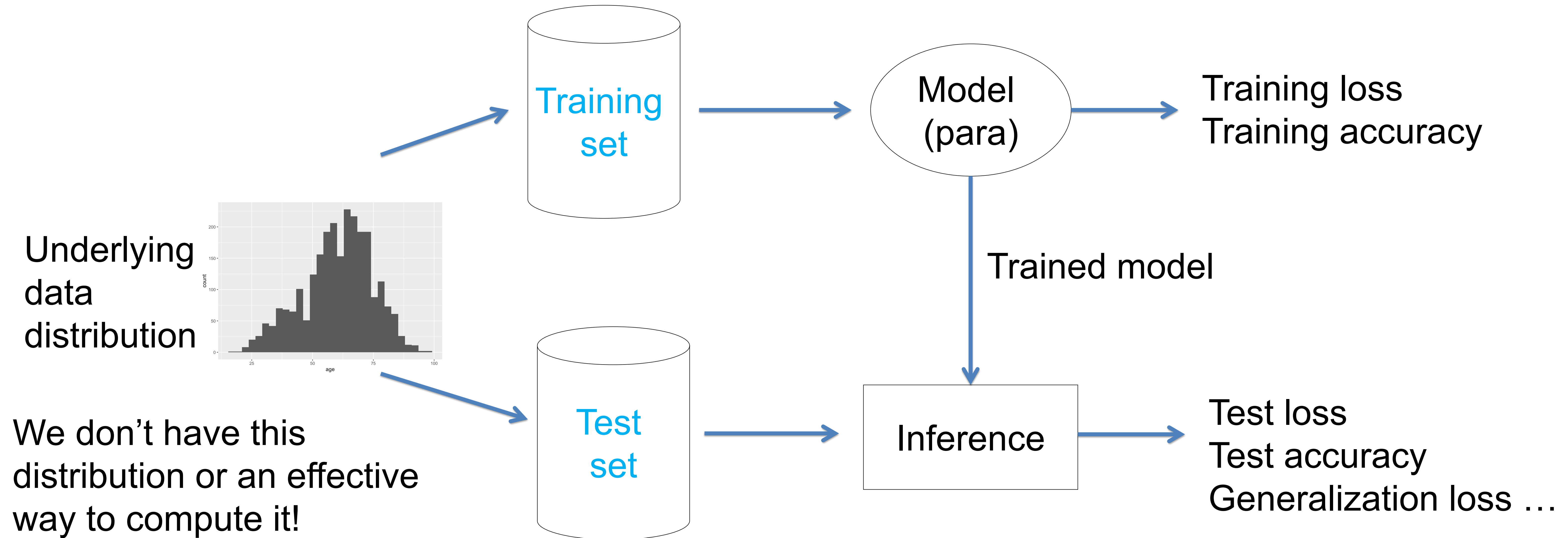We don't have this distribution or an effective way to compute it!
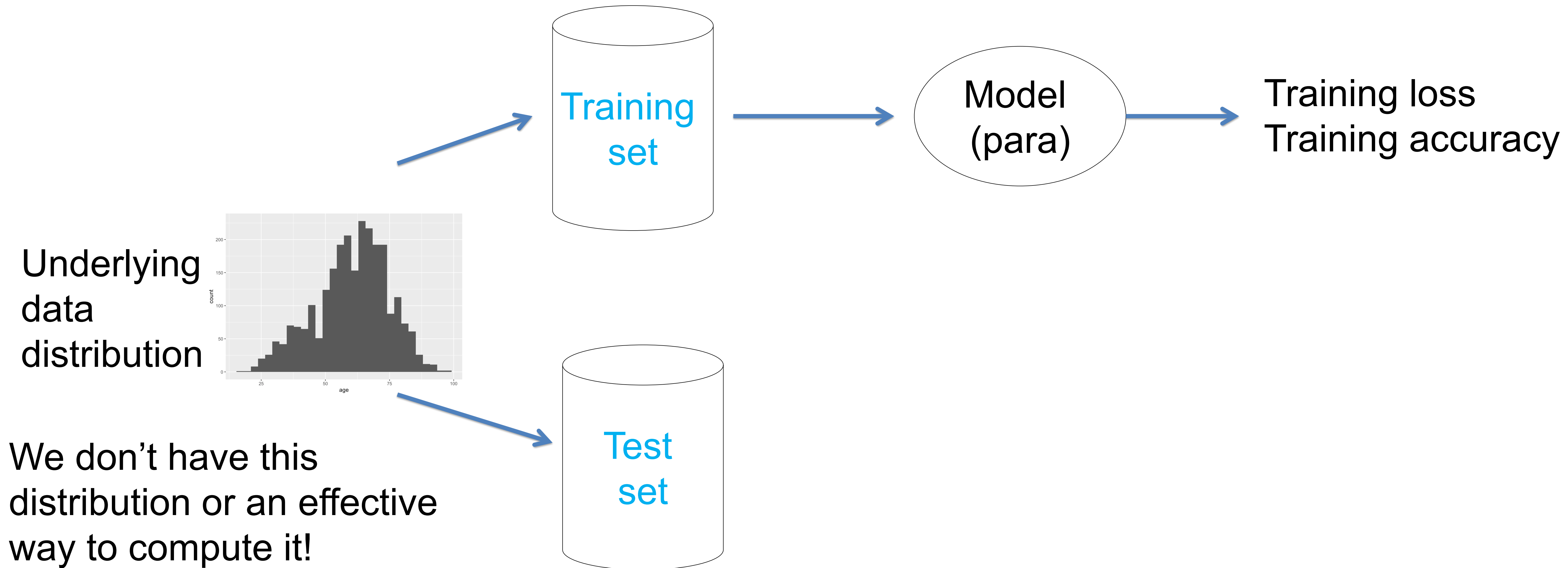
- Training and test sets are sampled from the same distribution, i.i.d. (independent and identically distributed)
- We care test/generalization performance; that is, the performance of training model on a new dataset

Model Generalization

- Training and test sets are sampled from the same distribution, i.i.d. (independent and identically distributed)
- We care test/generalization performance; that is, the performance of training model on a new dataset

Model Generalization

Underlying data distribution

We don't have this distribution or an effective way to compute it!

- Training and test sets are sampled from the same distribution, i.i.d. (independent and identically distributed)
- We care test/generalization performance; that is, the performance of training model on a new dataset

Model error

Given the underlying function $f(\boldsymbol{x})$, the model $M(\boldsymbol{x}; \boldsymbol{D})$ to approximate $f(x)$

$\boldsymbol{D}$ is a data set sampled from function $f(\boldsymbol{x})$. $\boldsymbol{D} = [(\boldsymbol{x_0}, \boldsymbol{y_0}), (\boldsymbol{x_1}, \boldsymbol{y_1}), \dots, (\boldsymbol{x_{N-1}}, \boldsymbol{y_{N-1}})]$

Every sample $(\boldsymbol{x_i}, \boldsymbol{y_i})$ is contaminated by noise: $\boldsymbol{y_i} = \boldsymbol{f}(\boldsymbol{x}) + \boldsymbol{\epsilon}$, $\boldsymbol{\epsilon}$ is the random noise

We want to know the expected error of model, given the dataset D:

$$E_{\boldsymbol{D}}[(y - M(\boldsymbol{x}; \boldsymbol{D}))^2]$$

This error consists of three parts:

$$E_{\boldsymbol{D}}\left[(y - M(\boldsymbol{x}; \boldsymbol{D}))^2\right] = \{E_{\boldsymbol{D}}[M(\boldsymbol{x}; \boldsymbol{D})] - f(\boldsymbol{x})\}^2 + E_{\boldsymbol{D}}\left[(E_{\boldsymbol{D'}}(M(\boldsymbol{x}; \boldsymbol{D'})) - M(\boldsymbol{x}; \boldsymbol{D}))^2\right] + \boldsymbol{\sigma}^2$$

21

Model error

Model prediction error consists of three parts:

$$E_{D}\left[(y - M(x; D))^2\right] = \{E_D[M(x; D)] - f(x)\}^2 + E_D\left[(E_D(M(x; D)) - M(x; D))^2\right] + \sigma^2$$

$$Bias(M, D) = E_D[M(x; D)] - f(x)$$

This is the **Bias**, for the difference between the mean model performance and ground-truth

$E_D[M(x; D)]$ is the expected model performance over all possible datasets <- the best model we can get

22

Model prediction error consists of three parts:

$$E_D\left[(y - M(\boldsymbol{x}; \boldsymbol{D}))^2\right] = \{E_D[M(\boldsymbol{x}; \boldsymbol{D})] - f(\boldsymbol{x})\}^2 + E_D\left[(E_{\boldsymbol{D'}}(M(\boldsymbol{x}; \boldsymbol{D'})) - M(\boldsymbol{x}; \boldsymbol{D}))^2\right] + \boldsymbol{\sigma}^2$$

$$Var(\boldsymbol{M}, \boldsymbol{D}) = E_D\left[\left[\, E_{\boldsymbol{D'}}(M(\boldsymbol{x}; \boldsymbol{D'})) - M(\boldsymbol{x}; \boldsymbol{D})\,\right]^2\right]$$

This is the **Variance,** measuring model performance fluctuation due to **different datasets**.

Measure how much the model prediction can change, after trained with different training sets

http://statweb.stanford.edu/~tibs/ElemStatLearn/

Model error

Model prediction error consists of three parts:

$$E_D\left[(y - M(\boldsymbol{x}; \boldsymbol{D}))^2\right] = \{E_D[M(\boldsymbol{x}; \boldsymbol{D})] - f(\boldsymbol{x})\}^2 + E_D\left[(E_{\boldsymbol{D}'}(M(\boldsymbol{x}; \boldsymbol{D}')) - M(\boldsymbol{x}; \boldsymbol{D}))^2\right] + \boldsymbol{\sigma}^2$$

$\boldsymbol{\sigma}^2$ Bayes error, irreducible error

lowest possible error rate for any classifier

If one would know exactly what process/distribution generates the data, one still cannot achieve 100% accuracy, due to randomness

If the test error has a lower-bound, there will be a Bias-Variance trade-off.
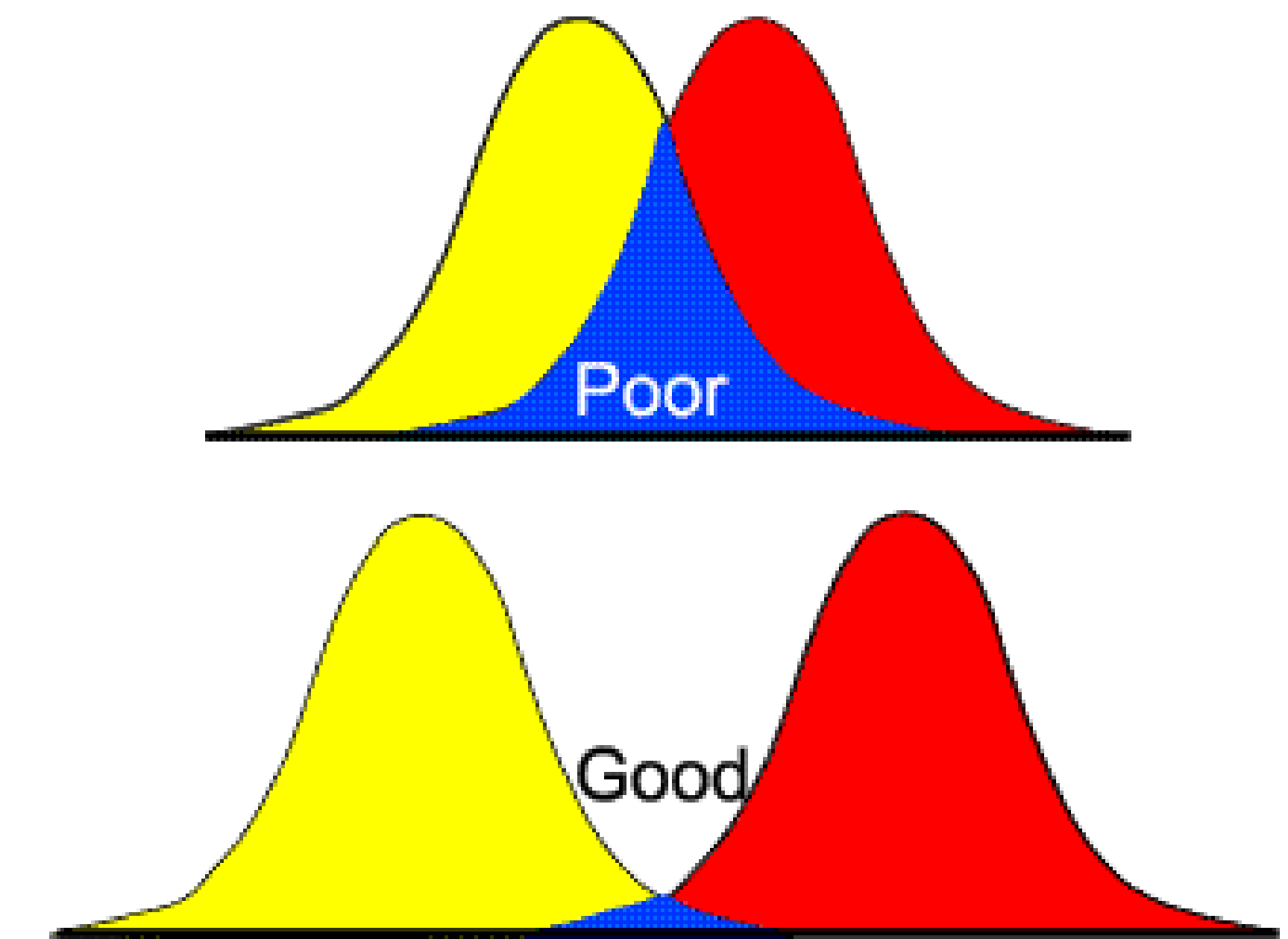
Distribution of two classes can overlap

Underfitting and overfitting

Polynomial fit degree 1
Training error: 0.4
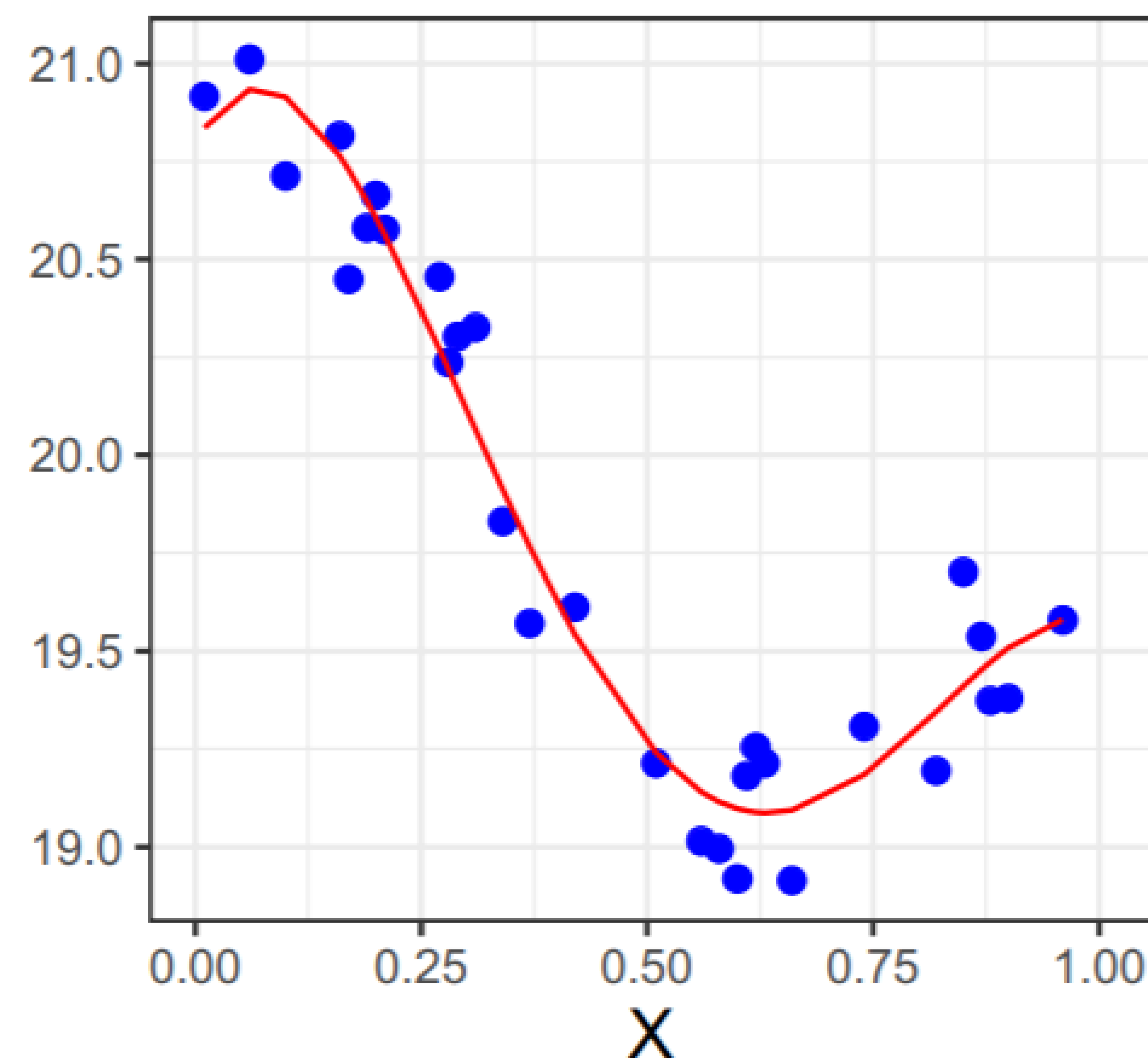Generalization error: 0.42
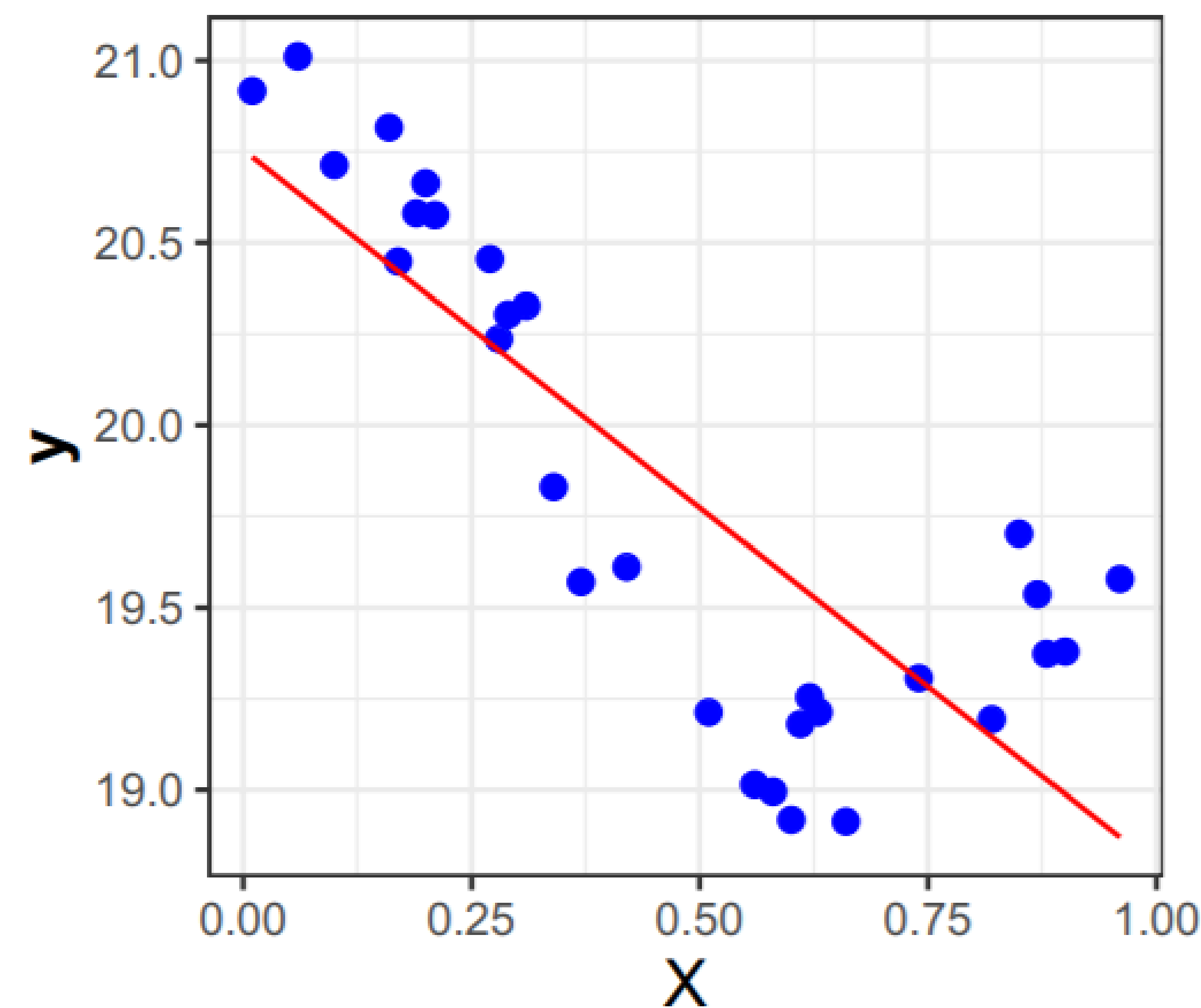
Underfit
Model cannot represent the data distribution

Polynomial fit degree 4
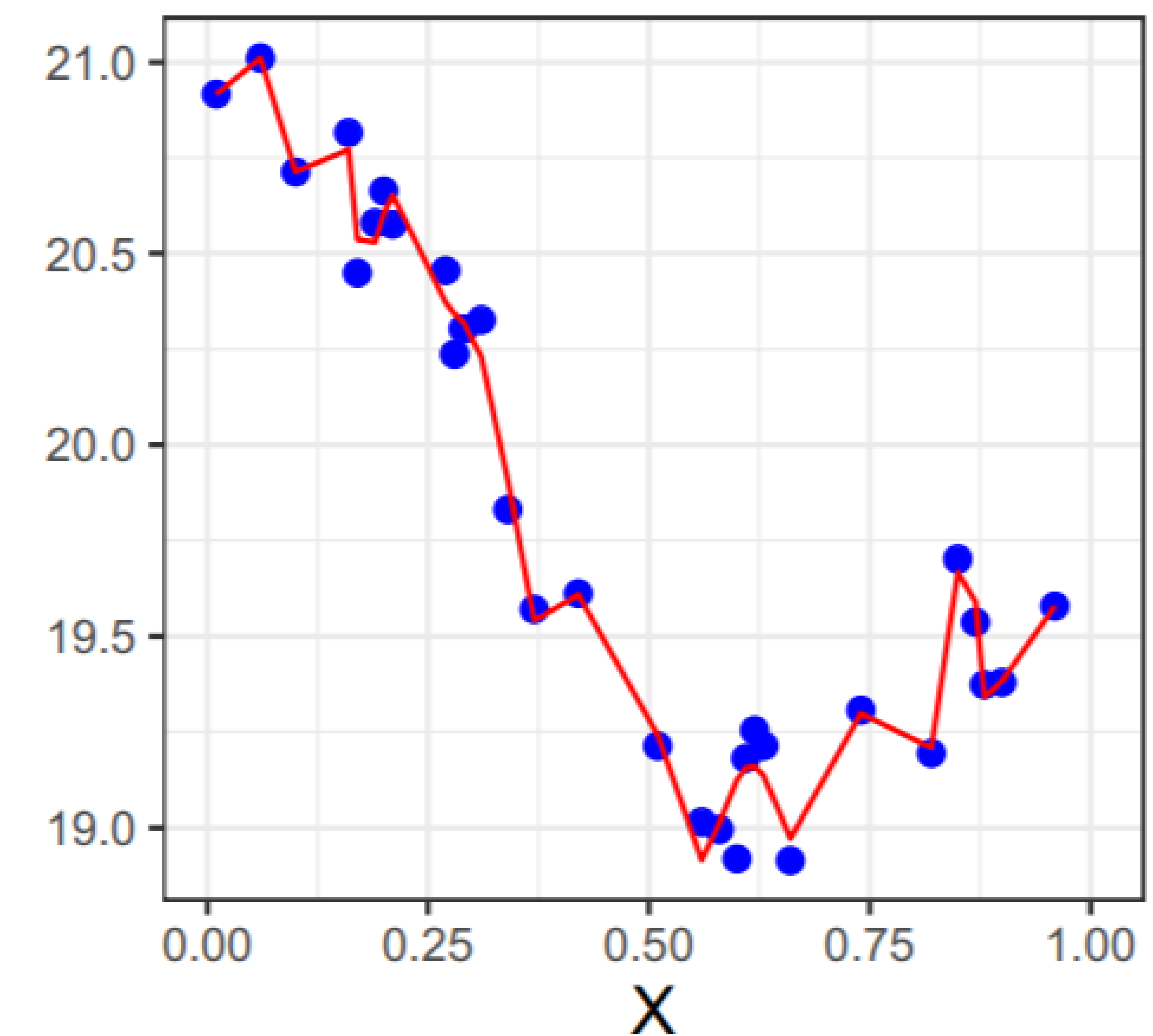Training error: 0.14
Generalization error: 0.17

Good fit
Model well represents the data distribution, and does not capture data noise

Polynomial fit degree 20
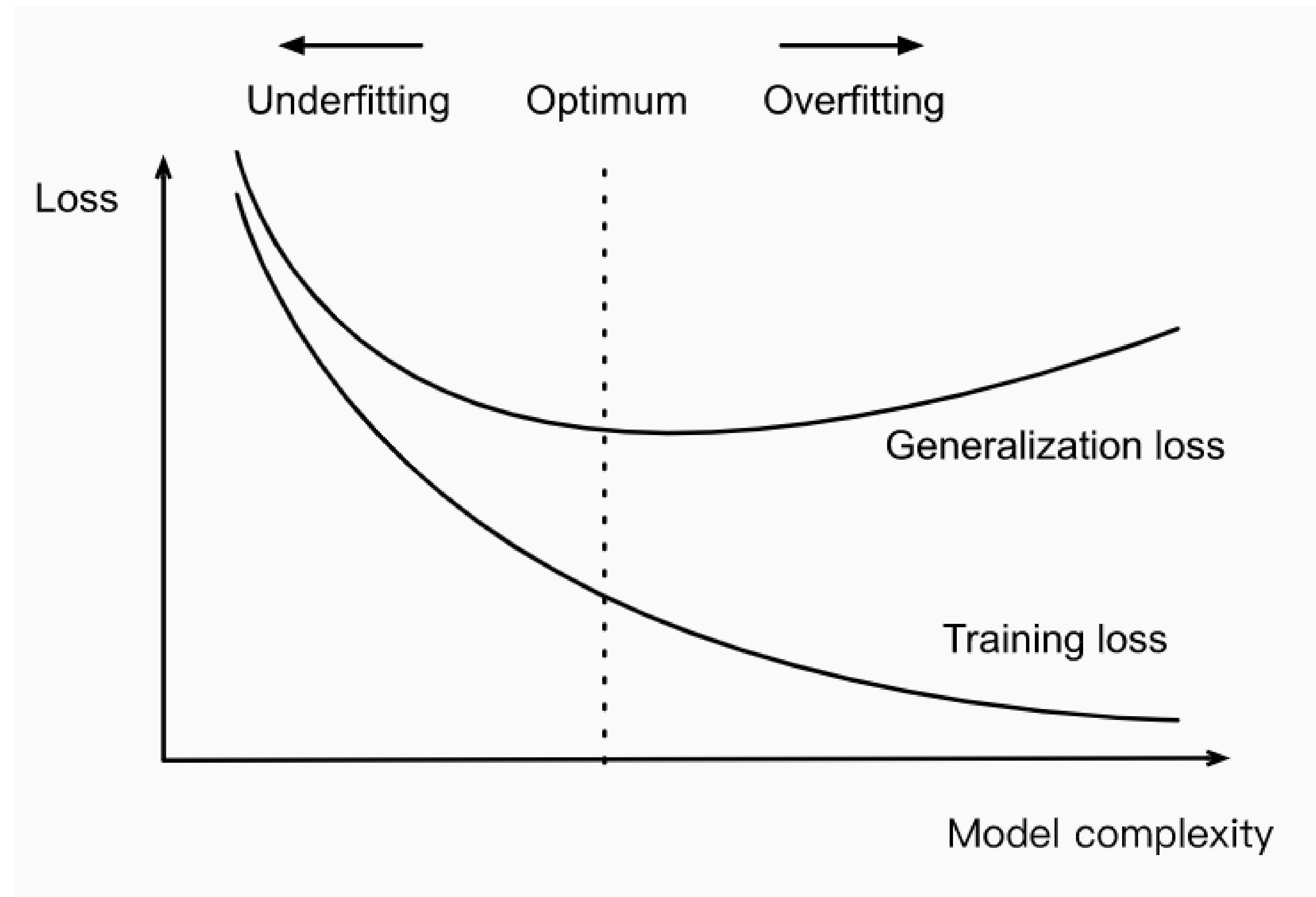Training error: 0.07
Generalization error: 2000

Overfit
Model is so flexible that it captures unwanted fluctuation due to noise

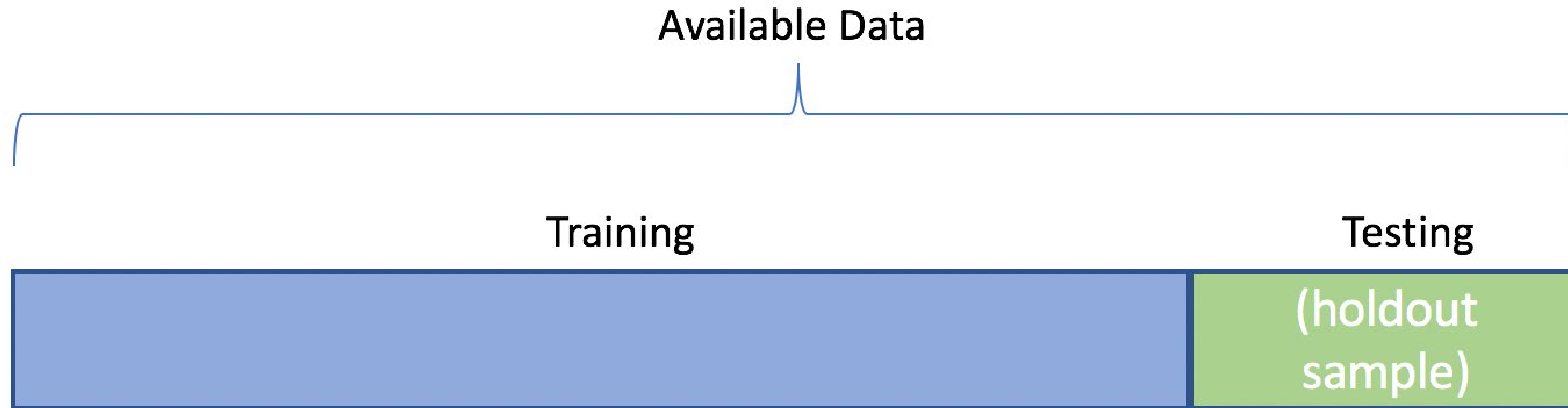https://ascpt.onlinelibrary.wiley.com/doi/10.1002/cpt.1796

Underfitting and overfitting

- Increasing model capacity/complexity can lead to overfitting

- When applying trained model to a new dataset, e.g. test set, model performance can decrease, as a result of overfitting, indicated by the high **generalization loss**

- Model can also underfitting the data, indicated by the high **training loss**

https://ja.d2l.ai/chapter_deep-learning-basics/underfit-overfit.html

In Deep Learning set up

Available Data

Training

Testing

(holdout sample)

- We train on Training set and use training accuracy to estimate model performance ← **Bias.**

- We apply the trained model on Testing set. Performance difference between training accuracy and test accuracy gives an estimation of **Variance**.

Bayes error is the best possible error rate if we knew true data distribution. It will not be zero :
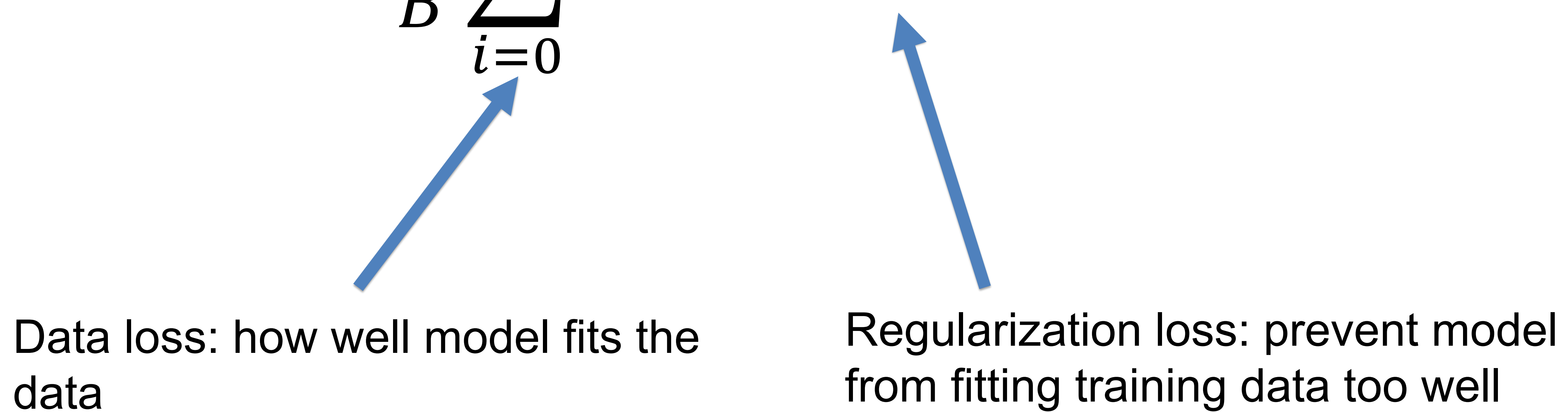- Overlap between class distribution
- Noise
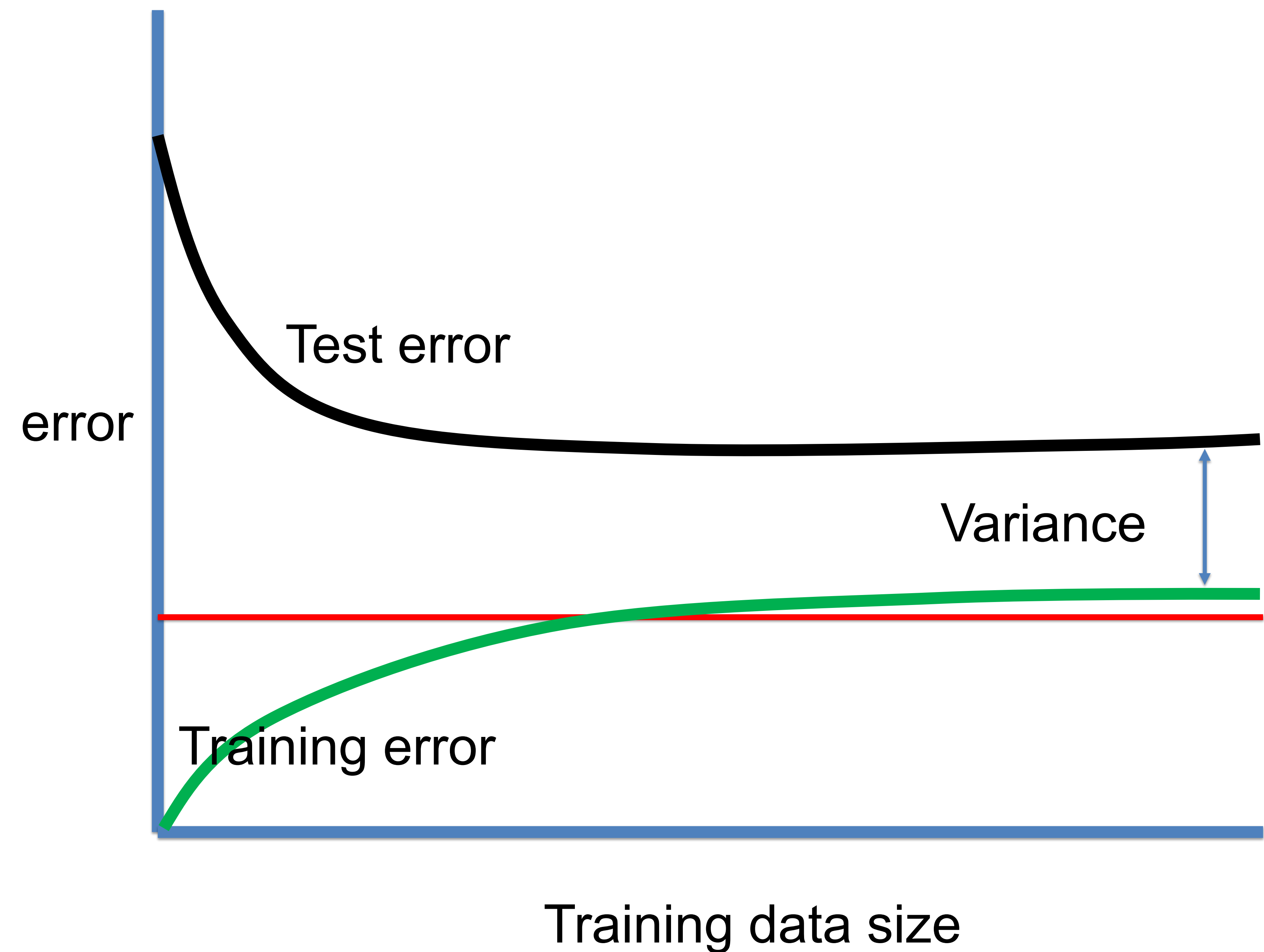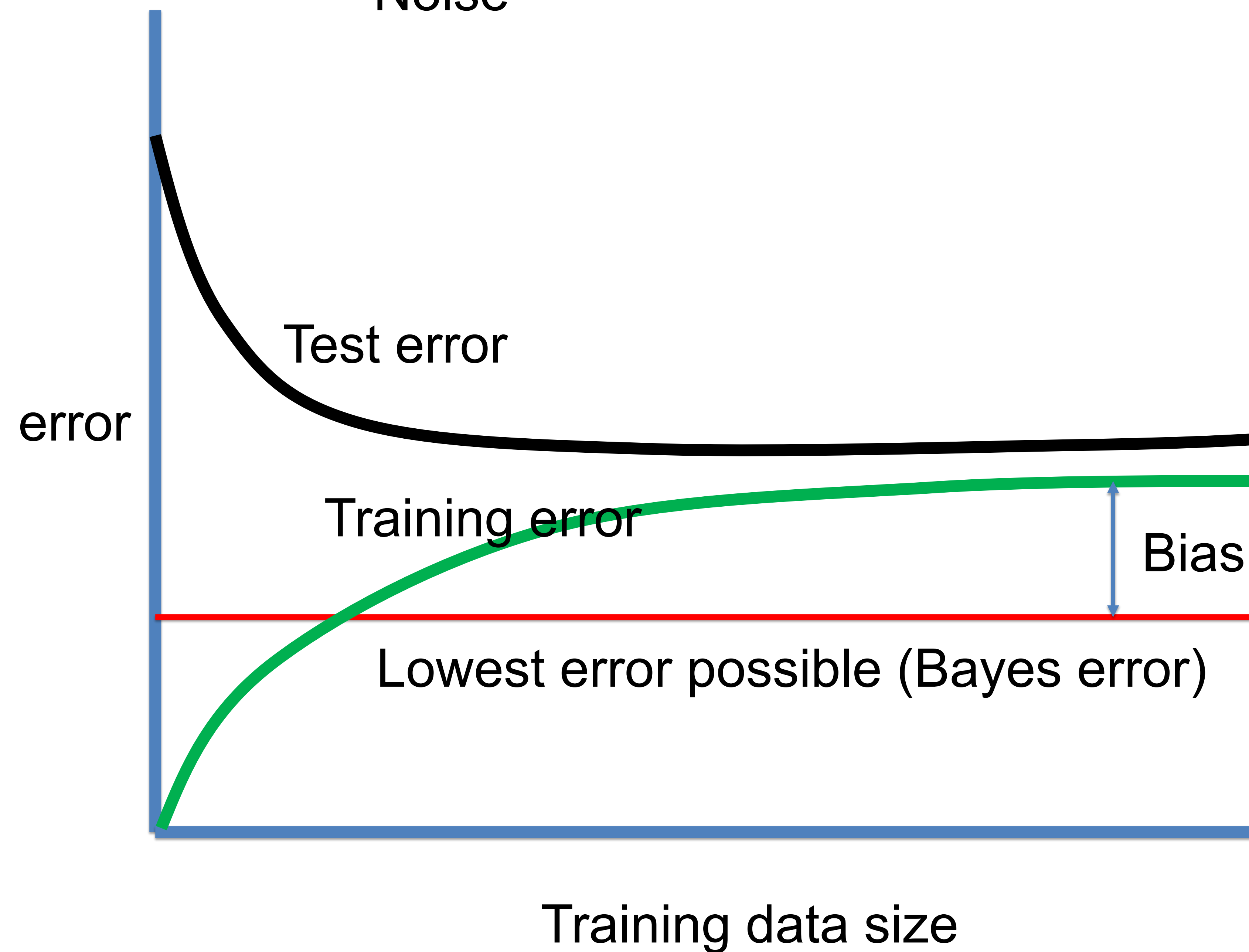
Regularization to control model complexity

Deep learning model is very powerful, that it may overfit training data → degraded generalization

IDEA: change loss function to control model complexity

$$L = \frac{1}{B} \sum_{i=0}^{B-1} L^{(i)} + \lambda R(\boldsymbol{W})$$

Data loss: how well model fits the data

Regularization loss: prevent model from fitting training data too well

L2 Regularization, weight decay

$$L = \frac{1}{B} \sum_{i=0}^{B-1} L^{(i)} + \frac{\lambda}{2} \|\boldsymbol{W}\|_2^2$$

$$\|\boldsymbol{W}\|_2^2 = \sum_{k=0}^{P-1} {w_k}^2$$

For all parameters in the model, flatten them and computing the element-wise L2 norm

$$\frac{\partial L}{\partial w_k} = \frac{1}{B} \sum_{i=0}^{B-1} \frac{\partial L^{(i)}}{\partial w_k} + \lambda w_k \qquad w_k = w_k - \alpha(\frac{1}{B} \sum_{i=0}^{B-1} \frac{\partial L^{(i)}}{\partial w_k}) - \alpha \lambda w_k$$

weight decay

L1 Regularization

$$L = \frac{1}{B} \sum_{i=0}^{B-1} L^{(i)} + \frac{\lambda}{2} \|\boldsymbol{W}\|_1$$

$$\|\boldsymbol{W}\|_1 = \sum_{k=0}^{P-1} |w_k|$$

For all parameters in the model, flatten them and computing the element-wise absolute value

$$\frac{\partial L}{\partial w_k} = \frac{1}{B} \sum_{i=0}^{B-1} \frac{\partial L^{(i)}}{\partial w_k} + \lambda sign(w_k) \qquad sign(w_k) = \begin{cases} 1, w_k > 0 \\ 0, w_k == 0 \\ -1, w_k < 0 \end{cases}$$

Drop Out

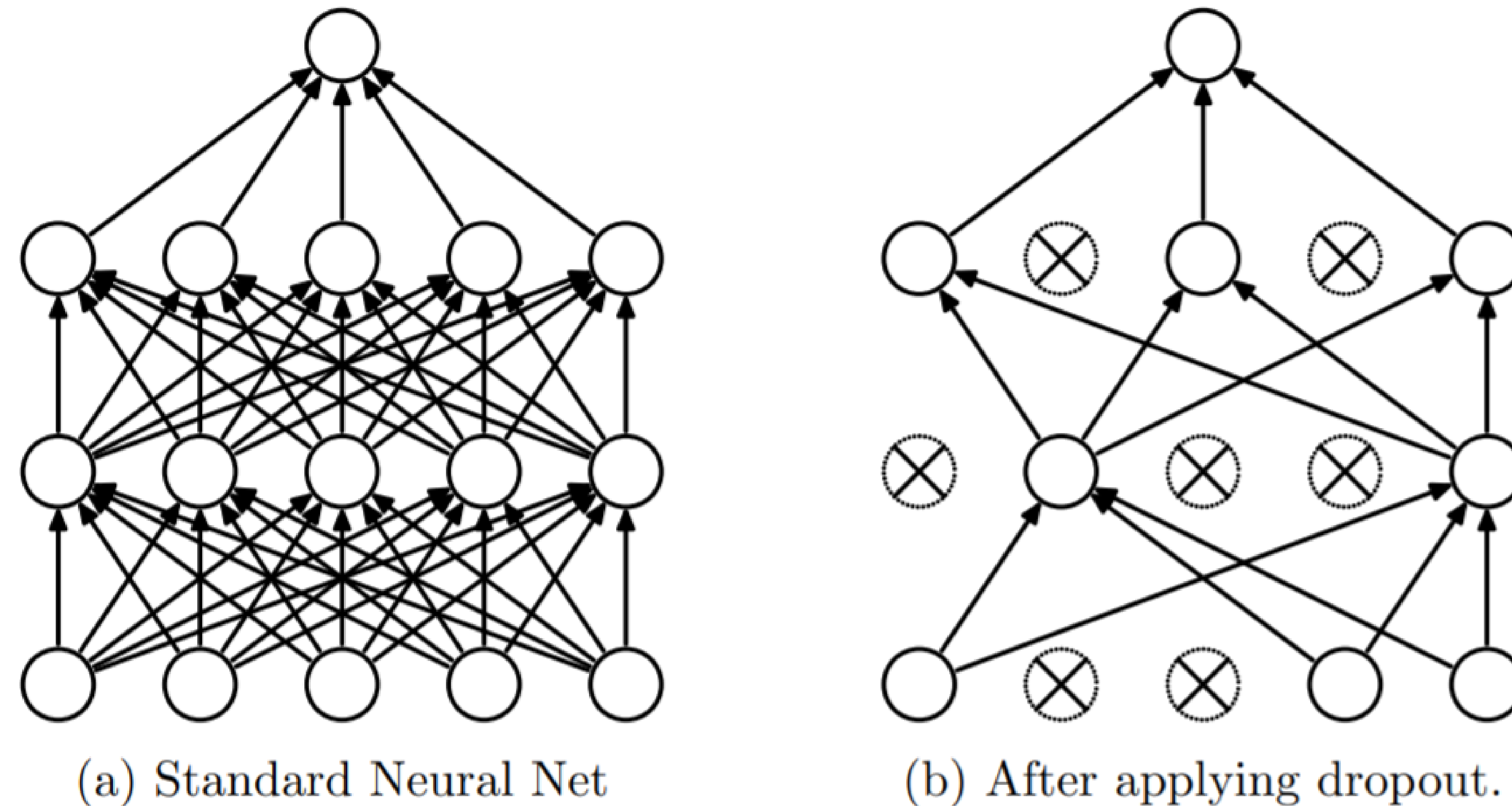(a) Standard Neural Net   (b) After applying dropout.

Figure 1: Dropout Neural Net Model. **Left**: A standard neural net with 2 hidden layers. **Right**: An example of a thinned net produced by applying dropout to the network on the left. Crossed units have been dropped.

- Jointly train many smaller network, randomly selected
- Implicitly combine exponentially many different neural network architectures efficiently

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. JMLR, 15(56):1929−1958, 2014.

- Randomly drop out a set of neurons during training phase, with a probability of 1-p (e.g. p=0.5)

- It means to randomly select different rows in the W matrix, for every batch, every epoch

- During the test time, use all neurons, but scale the score by p

- Or, in training time, scale the score by 1/p ← inverse dropout

- Often used with linear layer, not for convolution

- Require more epochs to train

32

Other operations with regularization effects

Regularization will:
- Increase training error
- Decrease testing error
- Introduce new hyper-parameters
- Often requires experiments

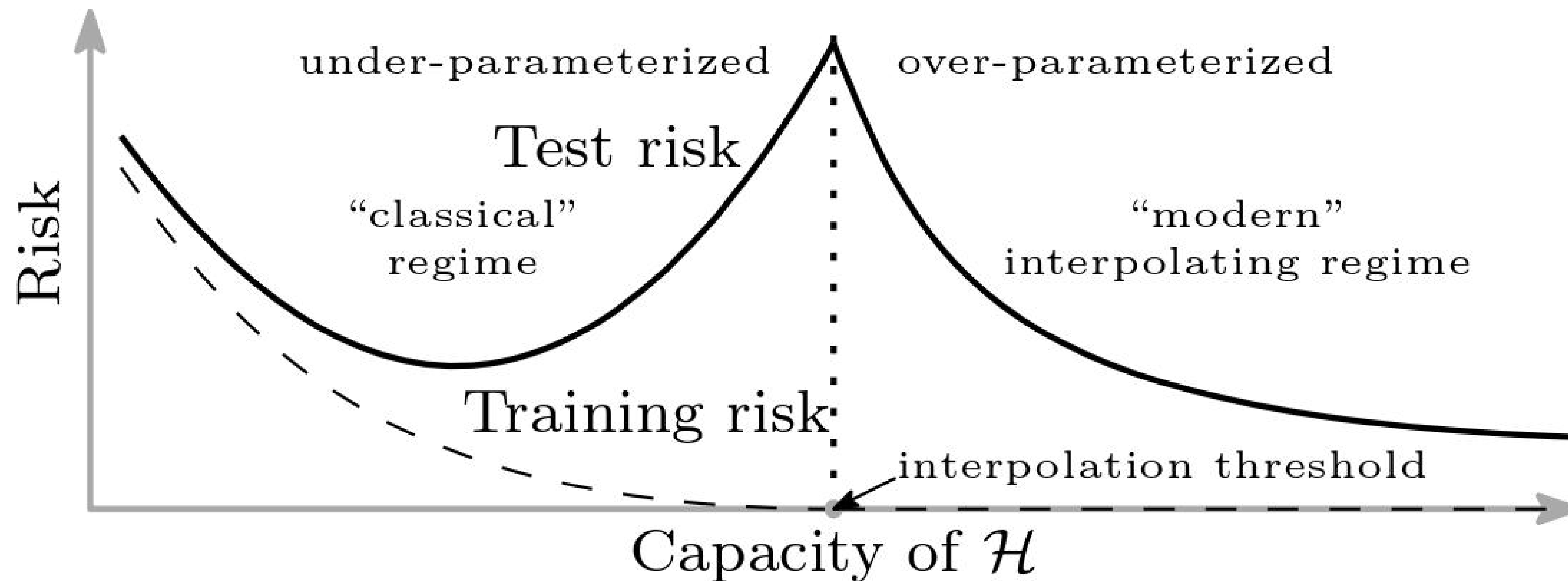Other operations to improve test error:

More on this topic in later lecturers

- Data augmentation
- Drop connection, random connection
- Batch/Layer/Channel normalization
- Early stopping
- …

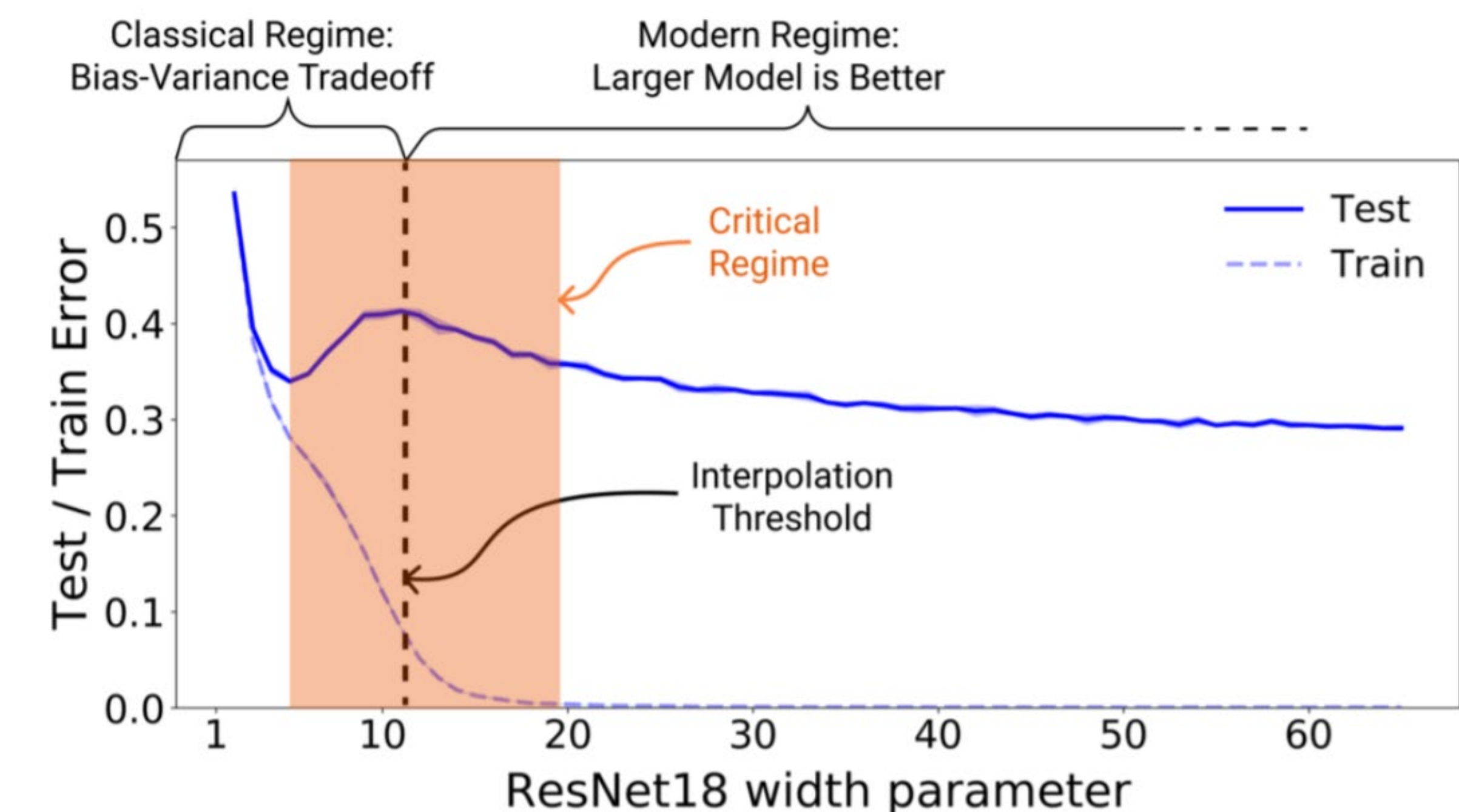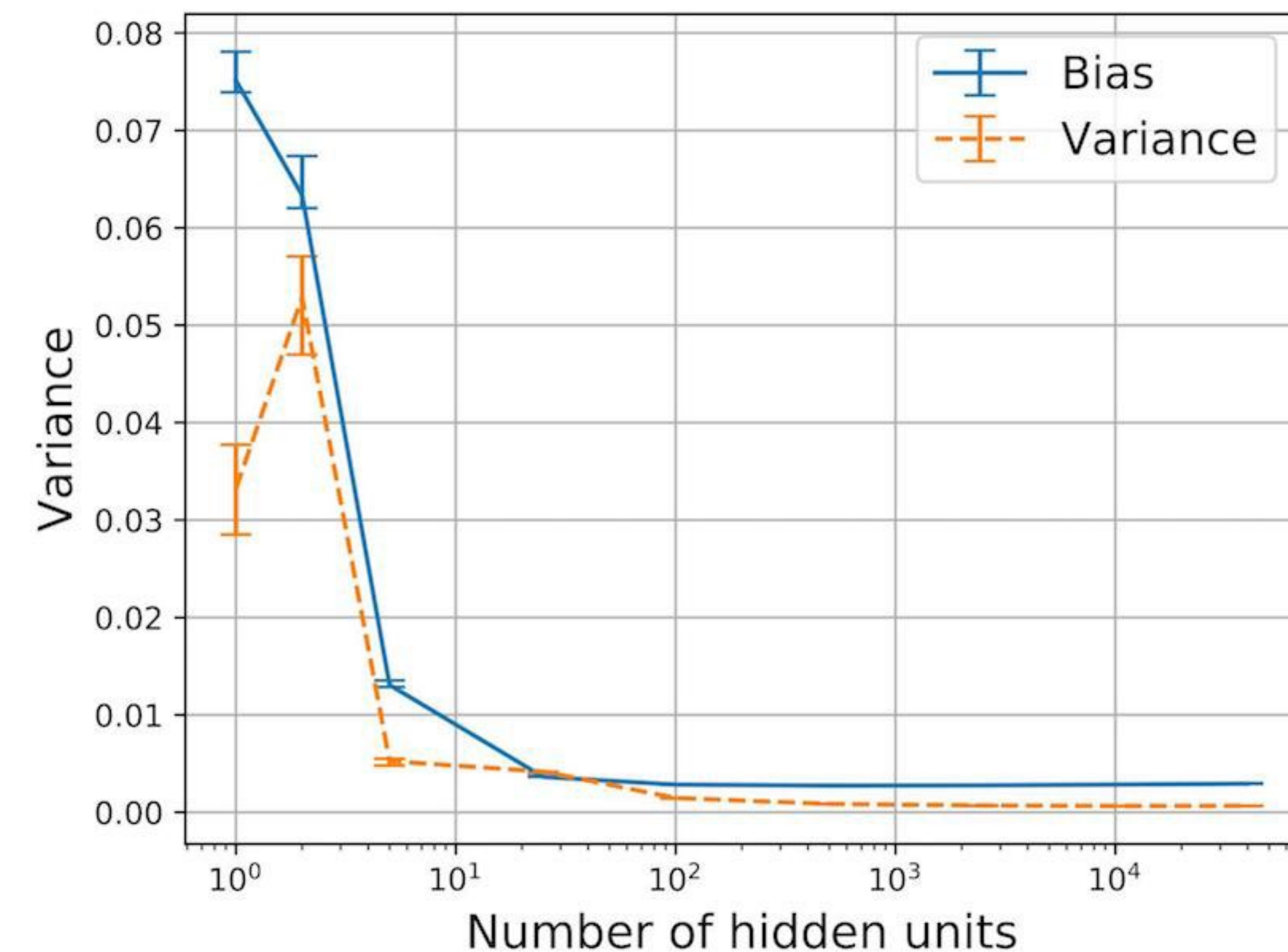Test error = Bias^2 + Variance + Bayes error

- Total test error can decrease with more complex model and longer training (#epochs)

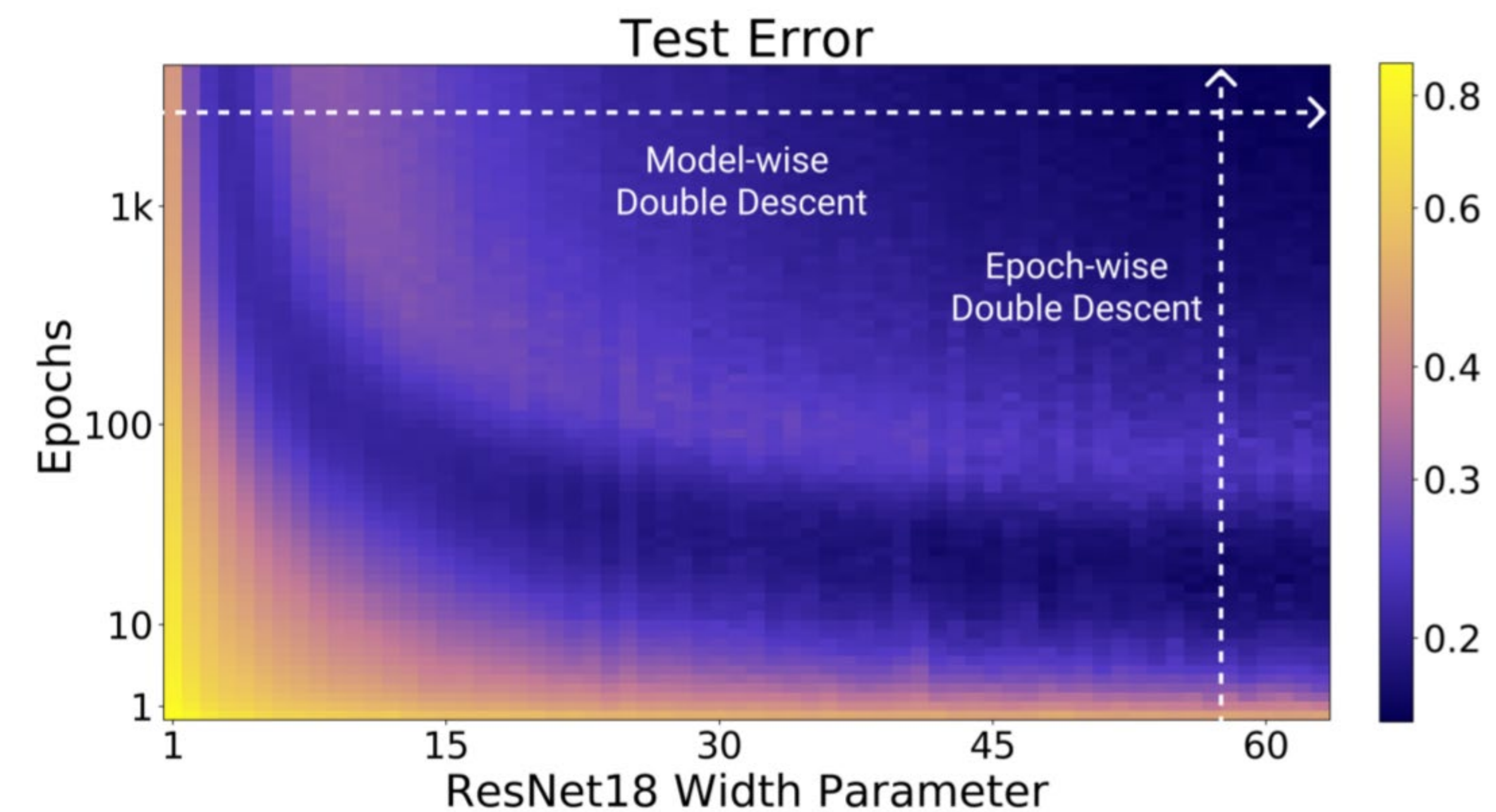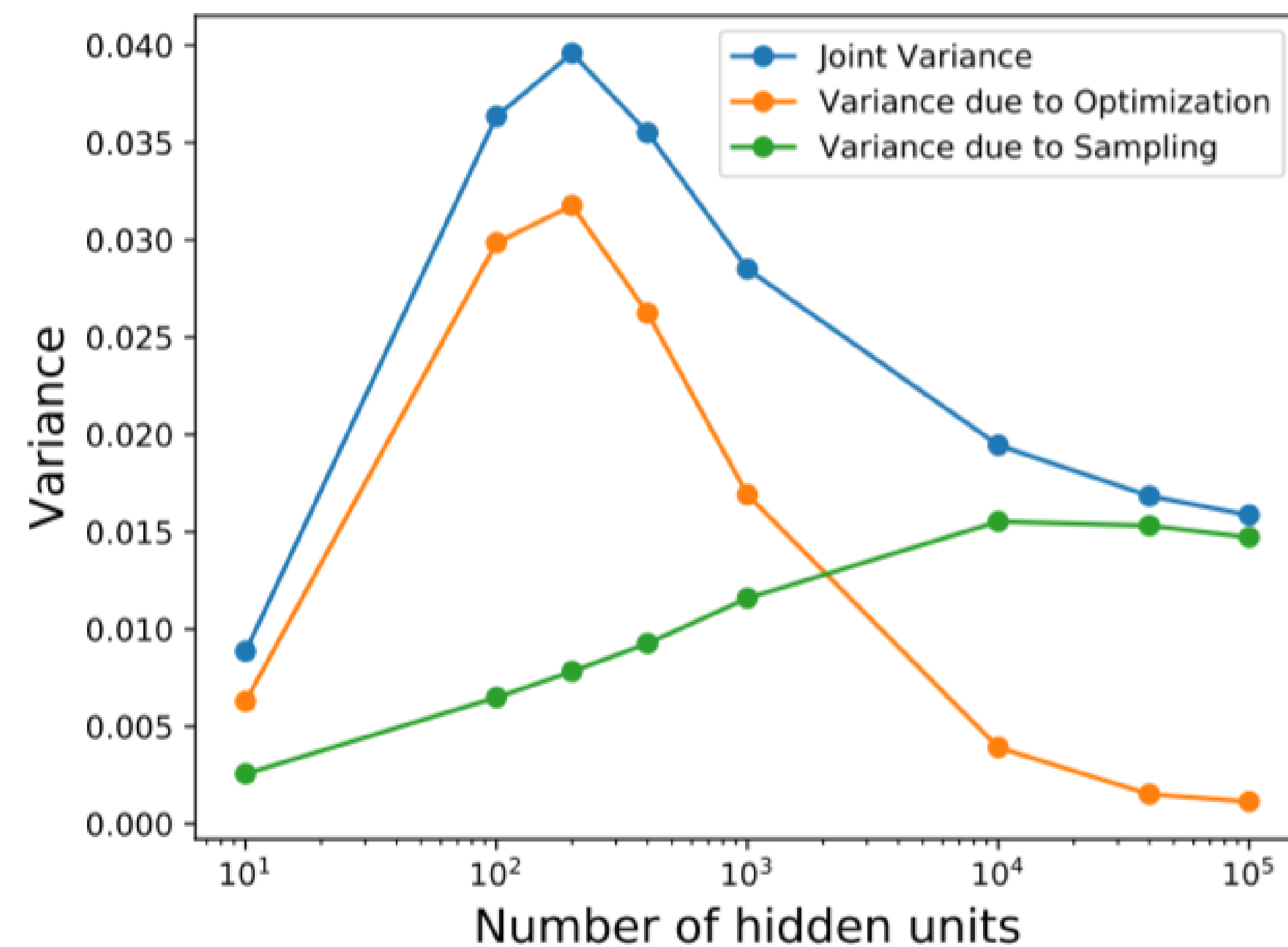- Still under research, but for deep model, Bias and variance may not act as a trade-off

https://arxiv.org/pdf/1912.08286.pdf
https://arxiv.org/pdf/1912.02292.pdf

34

Double-descent of test error in deep models

Test error = Bias^2 + Variance + Bayes error

$$Var(\boldsymbol{M}, \boldsymbol{D}) = E_{\boldsymbol{D}}\big[(E_{\boldsymbol{D'}}\big(M(\boldsymbol{x}; \boldsymbol{D'})\big) - M(\boldsymbol{x}; \boldsymbol{D}))^2\big]$$

- More complex model → reduce variance due to sampling the training set D; may help reduce variance due to optimization
- Longer training → reduce variance due to optimization

https://arxiv.org/pdf/1912.08286.pdf
https://arxiv.org/pdf/1912.02292.pdf

35