Deep Learning Crash Course

DLCC

www.deeplearningcrashcourse.org

Hui Xue

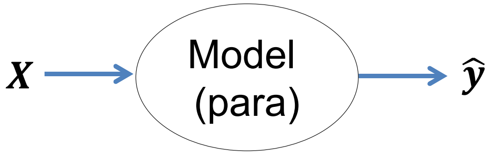
Fall 2021

Outline

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

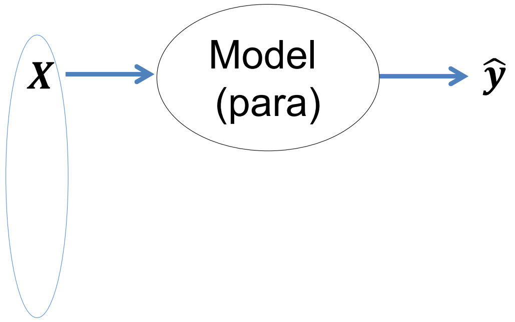
How do we know a model is good?

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



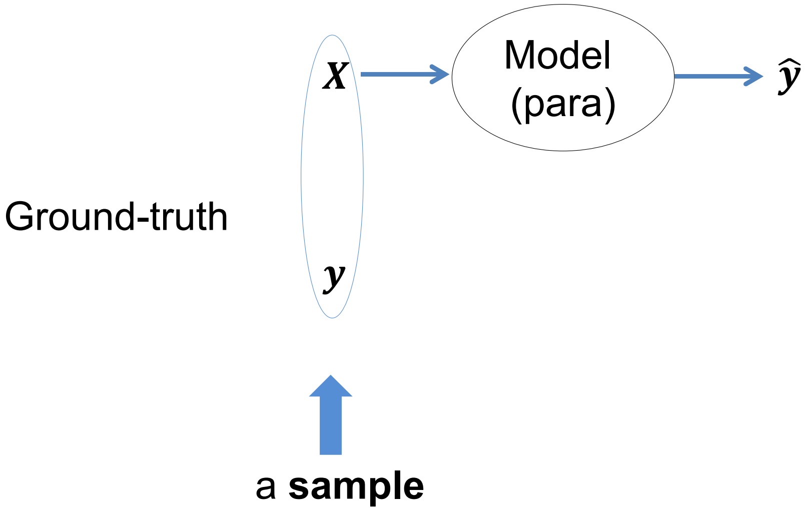
How do we know a model is good?

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



How do we know a model is good?

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



$\boldsymbol{X}$

Model (para)

$\hat{\boldsymbol{y}}$

Ground-truth

$\boldsymbol{y}$

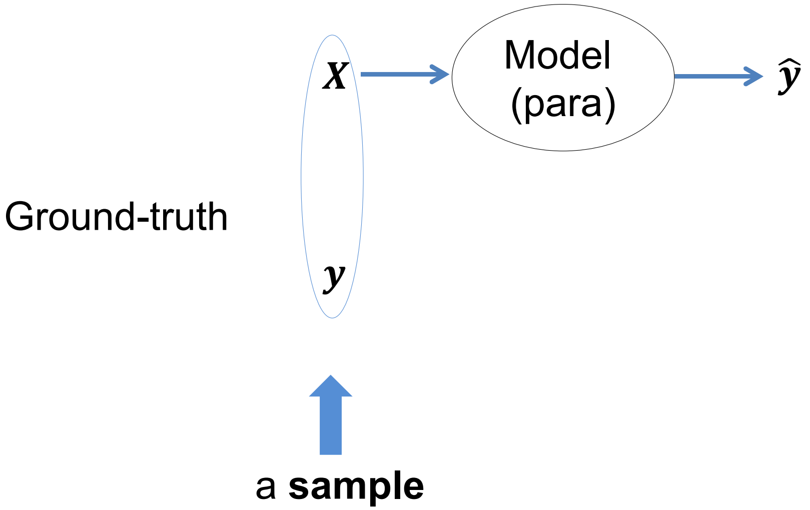
a **sample**

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

the i-th sample

How do we know a model is good?

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

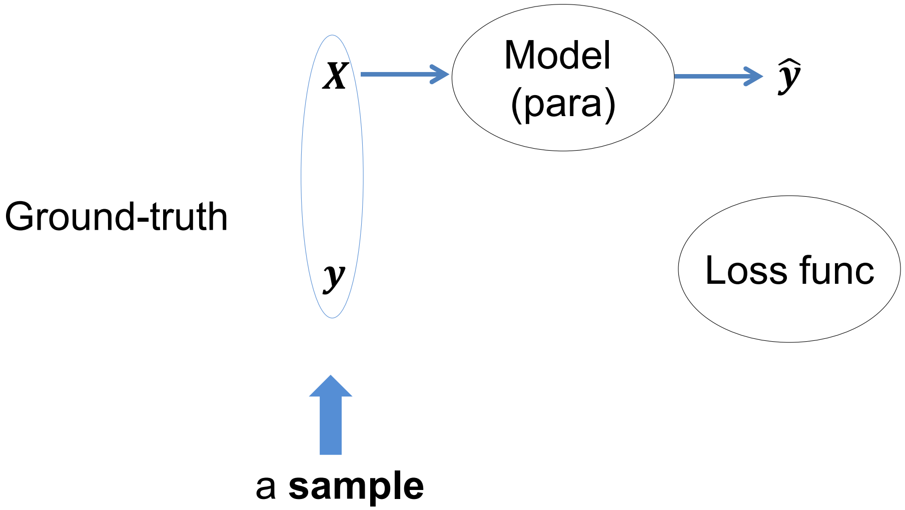


$\{\boldsymbol{X}^{(i)}, \boldsymbol{y}^{(i)}\}$

the i-th sample

How do we know a model is good?

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



$\boldsymbol{X}$

Model (para)

$\hat{\boldsymbol{y}}$

Ground-truth

$\boldsymbol{y}$

Loss func

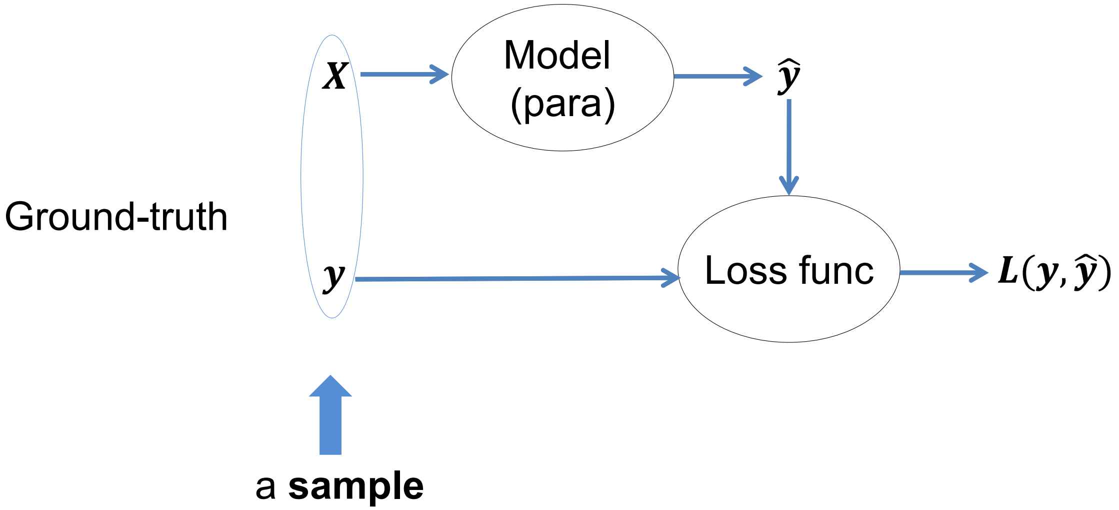
a **sample**

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

the i-th sample

How do we know a model is good?

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



Ground-truth

$\boldsymbol{X}$

$\boldsymbol{y}$

Model (para)

$\hat{\boldsymbol{y}}$

Loss func

$\boldsymbol{L}(\boldsymbol{y}, \hat{\boldsymbol{y}})$

a **sample**

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

the i-th sample

Cross-entropy

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

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

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

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

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

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

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

$\boldsymbol{H(p)}$ is the entropy of distribution p

$$\boldsymbol{H(p,q) \neq H(q,p)}$$

$$\begin{aligned}
\boldsymbol{H(p,q)} &= -\sum_{\boldsymbol{x}} \boldsymbol{p(x)} \log[\boldsymbol{q(x)}] \\
&= -\sum_{\boldsymbol{x}} \boldsymbol{p(x)} \log\left[\frac{\boldsymbol{p(x)}}{\boldsymbol{p(x)}} \boldsymbol{q(x)}\right] \\
&= -\sum_{\boldsymbol{x}} \boldsymbol{p(x)} \left\{\log[\boldsymbol{p(x)}] + \log\left[\frac{\boldsymbol{q(x)}}{\boldsymbol{p(x)}}\right]\right\} \\
&= -\sum_{\boldsymbol{x}} \boldsymbol{p(x)} \log[\boldsymbol{p(x)}] - \sum_{\boldsymbol{x}} \boldsymbol{p(x)} \log\left[\frac{\boldsymbol{q(x)}}{\boldsymbol{p(x)}}\right] \\
&= \underbrace{-\sum_{\boldsymbol{x}} \boldsymbol{p(x)} \log[\boldsymbol{p(x)}]}_{\boldsymbol{H(p)}} + \underbrace{\sum_{\boldsymbol{x}} \boldsymbol{p(x)} \log\left[\frac{\boldsymbol{p(x)}}{\boldsymbol{q(x)}}\right]}_{\boldsymbol{D_{KL}(p \parallel q)}}
\end{aligned}$$

Entropy of distribution p

Binary Cross-entropy loss

For the binary classification,

$\boldsymbol{y}$ = 0 for the “Yes” class and 1 for the “No” class

$\widehat{\boldsymbol{y}} \in [\mathbf{0}, \mathbf{1}]$, probability belonging to the “Yes” class
$\mathbf{1} - \widehat{\boldsymbol{y}}$ for the “No” class

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

$$\boldsymbol{L}_{\boldsymbol{BCE\_loss}}(\boldsymbol{y}, \widehat{\boldsymbol{y}}) = -[\boldsymbol{y} \cdot \boldsymbol{log}(\widehat{\boldsymbol{y}}) + (\mathbf{1} - \boldsymbol{y}) \cdot \boldsymbol{log}(\mathbf{1} - \widehat{\boldsymbol{y}})]$$

To minimize the BCE loss,

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

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

For all other cases, $\boldsymbol{L}_{\boldsymbol{BCE\_loss}}$ > 0

Cross-entropy loss for multi-class

For the multi-class classification,

$$\boldsymbol{y} = \left\{ \begin{array}{lccccc} \boldsymbol{CH3}, & \mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \boldsymbol{CH2}, & \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \boldsymbol{CH4}, & \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{0} \\ \boldsymbol{SAX}, & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{0} \\ \boldsymbol{Other}, & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{1} \end{array} \right.$$

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 $\boldsymbol{y}$

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

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

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

Cross-entropy loss

For the multi-class classification,

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

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

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

y_k is the index of correct class for sample $\boldsymbol{y}$

y is one-hot encoding for the correct class

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

Only one term left!

To minimize the cross-entropy loss,

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

Cross-entropy loss

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

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

$$= -[0 \times \log(0.2)$$

$$+0 \times \log(0.3)$$

$$+1 \times \log(0.1)$$

$$+0 \times \log(0.15)$$

$$+0 \times \log(0.25)]$$

$$= -\log(0.1) = 2.3$$

Cross-entropy loss

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

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

$$= -[0 \times \log(0.02)$$

$$+0 \times \log(0.1)$$

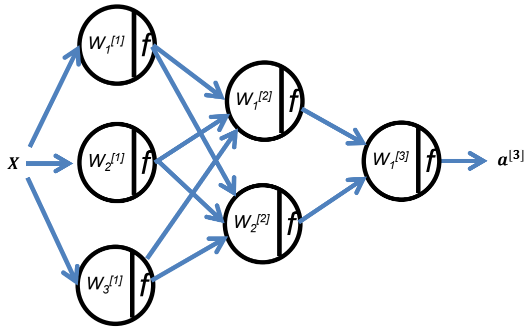
$$+1 \times \log(0.8)$$

$$+0 \times \log(0.05)$$

$$+0 \times \log(0.03)]$$

$$= -\log(0.8) = 0.22$$

Where we are



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

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

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

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

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

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

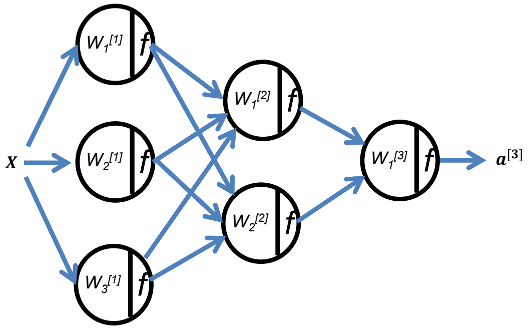
$$\boldsymbol{L}_{\boldsymbol{BCE\_loss}}(\boldsymbol{y}, \hat{\boldsymbol{y}})$$

$$\boldsymbol{L}_{\boldsymbol{CE\_loss}}(\boldsymbol{y}, \hat{\boldsymbol{y}})$$

Model : map input to output

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

How do we determine the model parameters?

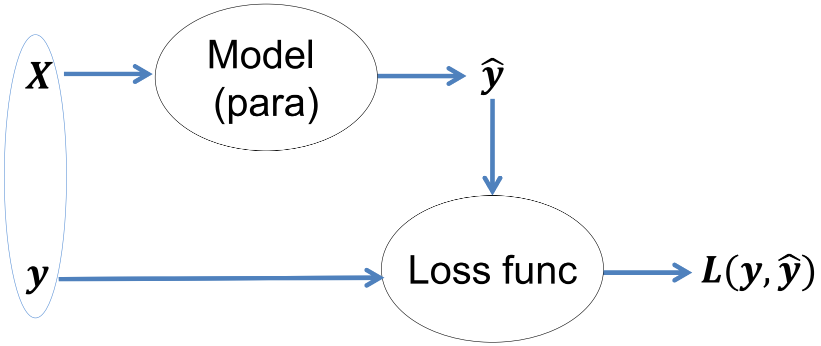


$\boldsymbol{W}^{[1]}, \boldsymbol{b}^{[1]}$

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

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

Model : map input to output



$\boldsymbol{X}$

Model (para)

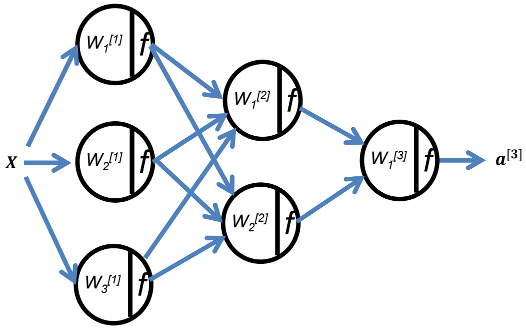
$\hat{\boldsymbol{y}}$

$\boldsymbol{y}$

Loss func

$\boldsymbol{L}(\boldsymbol{y}, \hat{\boldsymbol{y}})$

How do we determine the model parameters?

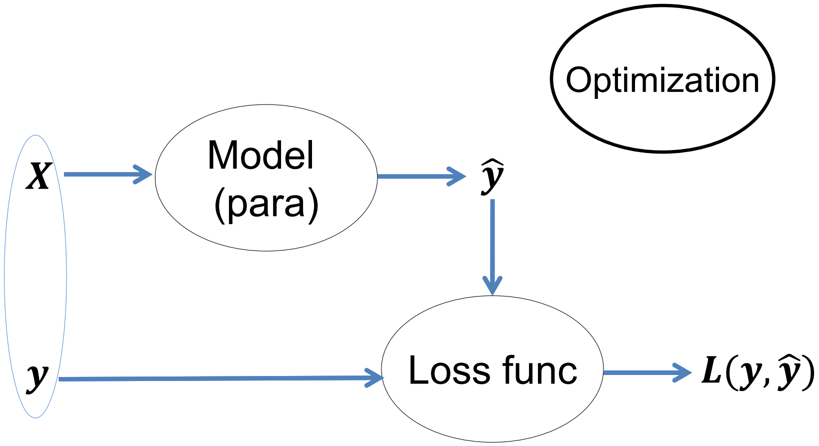


$\boldsymbol{W}^{[1]}, \boldsymbol{b}^{[1]}$

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

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

Model : map input to output



Optimization

$\boldsymbol{X}$

Model (para)

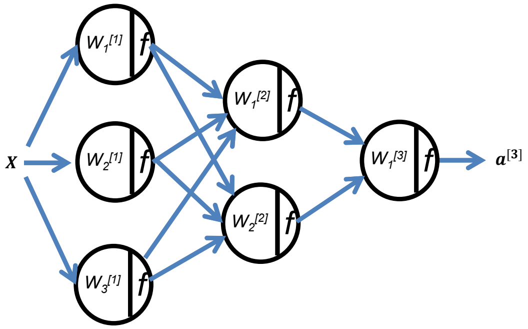
$\hat{\boldsymbol{y}}$

$\boldsymbol{y}$

Loss func

$\boldsymbol{L}(\boldsymbol{y}, \hat{\boldsymbol{y}})$

How do we determine the model parameters?

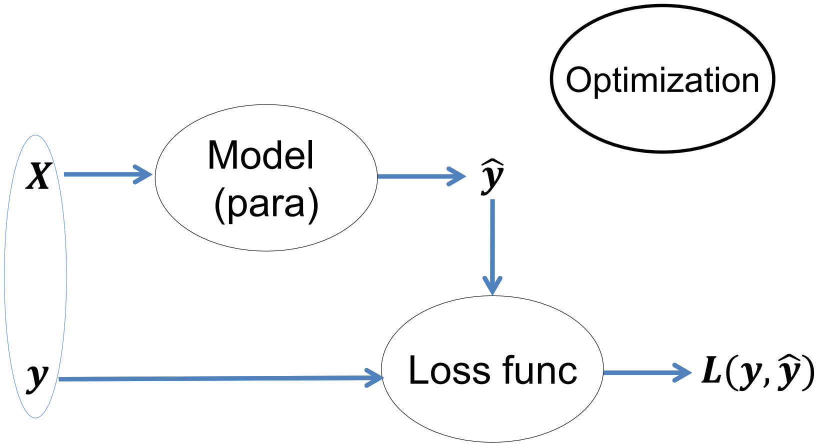


$\boldsymbol{W}^{[1]}, \boldsymbol{b}^{[1]}$

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

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

Model : map input to output



Optimization

$\boldsymbol{X}$

Model (para)

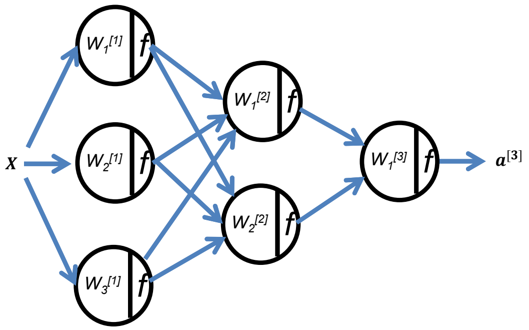
$\widehat{\boldsymbol{y}}$

$\boldsymbol{y}$

Loss func

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

How do we determine the model parameters?

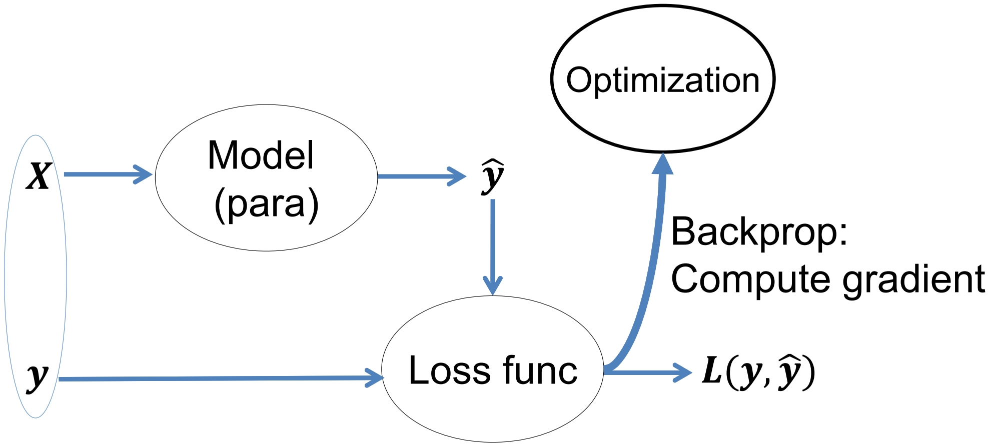


$\boldsymbol{W}^{[1]}, \boldsymbol{b}^{[1]}$

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

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

Model : map input to output



$\boldsymbol{X}$

Model (para)

$\widehat{\boldsymbol{y}}$

Optimization

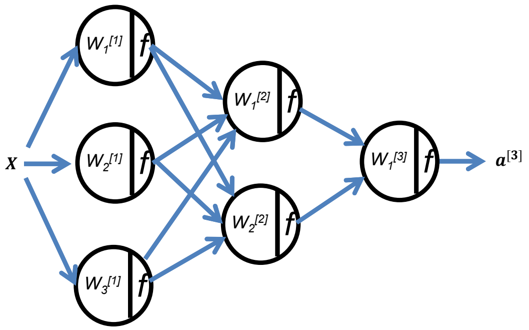
Backprop:
Compute gradient

$\boldsymbol{y}$

Loss func

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

How do we determine the model parameters?

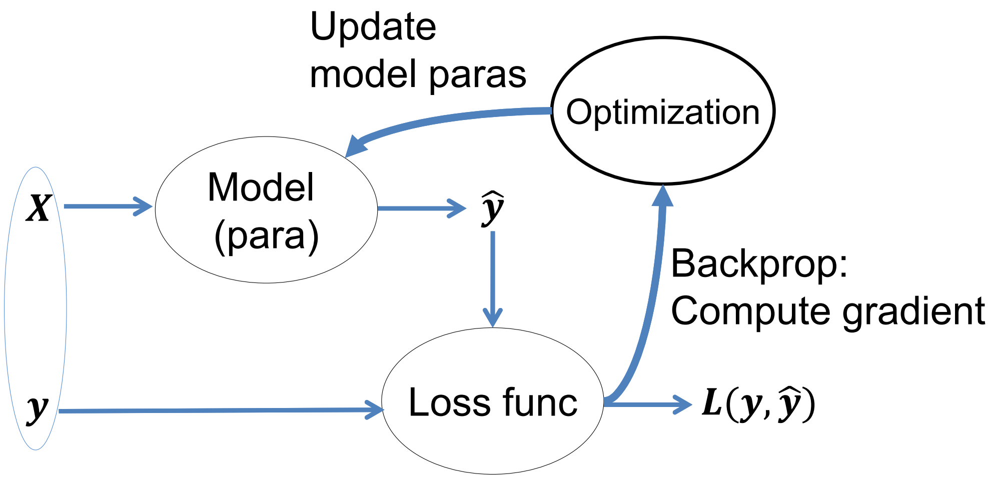


$\boldsymbol{W}^{[1]}, \boldsymbol{b}^{[1]}$

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

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

Model : map input to output



Update
model paras

Optimization

$\boldsymbol{X}$

Model
(para)

$\hat{\boldsymbol{y}}$

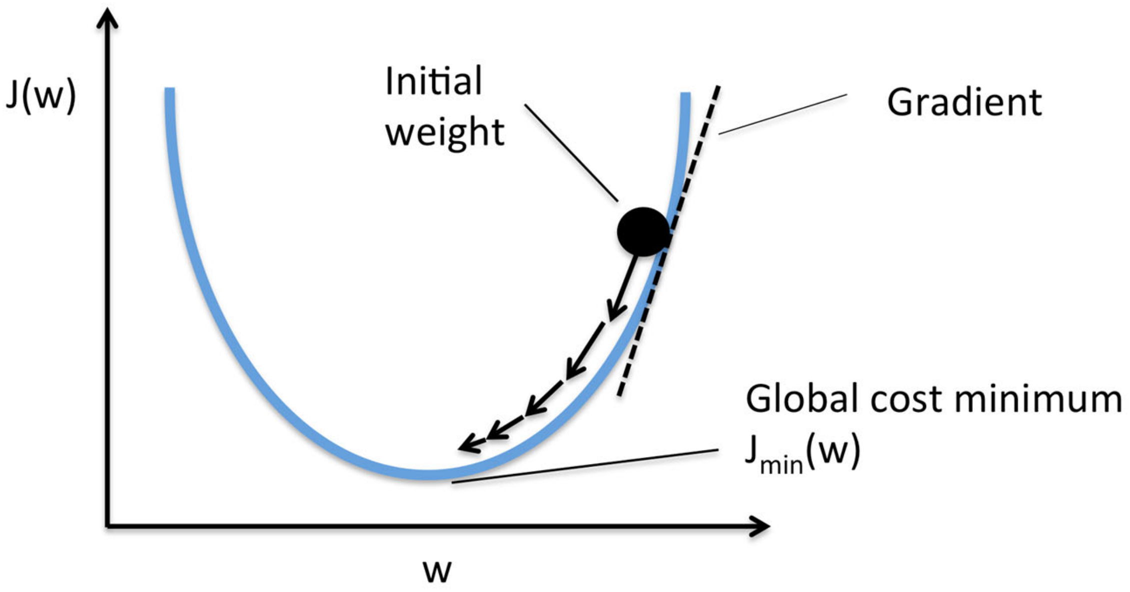
Backprop:
Compute gradient

$\boldsymbol{y}$

Loss func

$\boldsymbol{L}(\boldsymbol{y}, \hat{\boldsymbol{y}})$

Gradient descent



To fine minimal point of a function

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

https://www.quora.com/Whats-the-difference-between-gradient-descent-and-stochastic-gradient-descent

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.

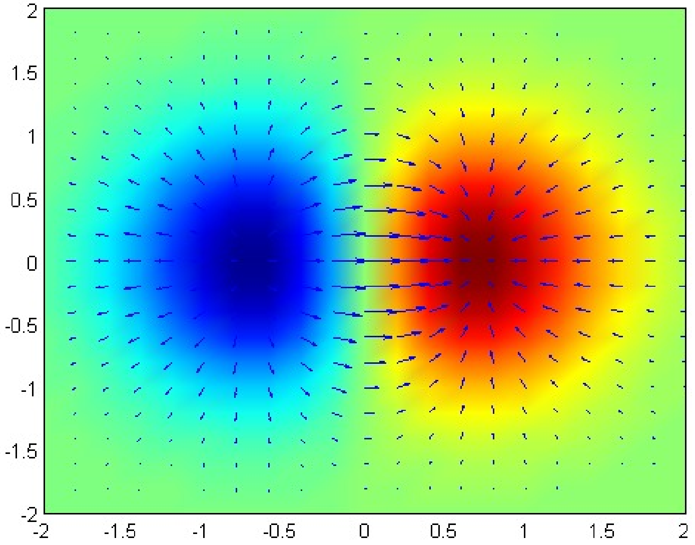
In 1-dimension, the derivative of a function:

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

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

- $\boldsymbol{p} = [\boldsymbol{x_0}, \boldsymbol{x_1}, \boldsymbol{x_2}, \dots, \boldsymbol{x_{N-1}}]^{\boldsymbol{T}}$
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 $|\nabla f(\boldsymbol{p})|$

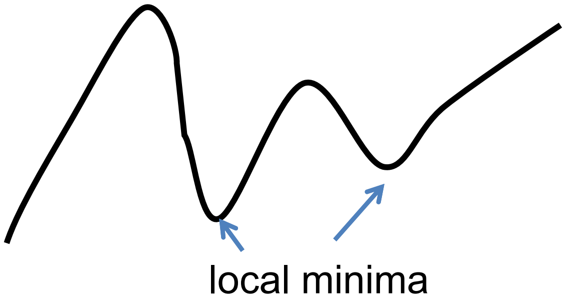


https://en.wikipedia.org/wiki/Gradient#/media/File:Gradient_of_a_Function.tif

Gradient descent

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

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



For every layer l :

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

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

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 = $\frac{1}{M}\sum_{i=0}^{M-1} L(\boldsymbol{y}^{(\boldsymbol{i})}, \widehat{\boldsymbol{y}}^{(\boldsymbol{i})}(\boldsymbol{W}, \boldsymbol{b}))$

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

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

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

Gradient descent

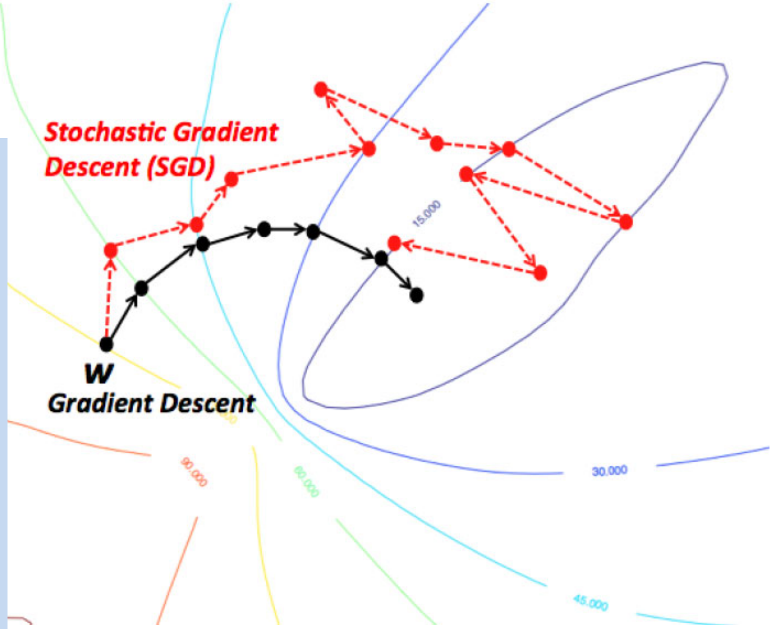
```
Initialize weights and bias

for iter in range(t):

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

    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
        $L = L(\boldsymbol{y}^{(\boldsymbol{i})}, \widehat{\boldsymbol{y}}^{(\boldsymbol{i})}(\boldsymbol{W}, \boldsymbol{b}))$

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

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



https://data-notes.co/the-10-deep-learning-methods-ai-practitioners-need-to-apply-885259f402c1

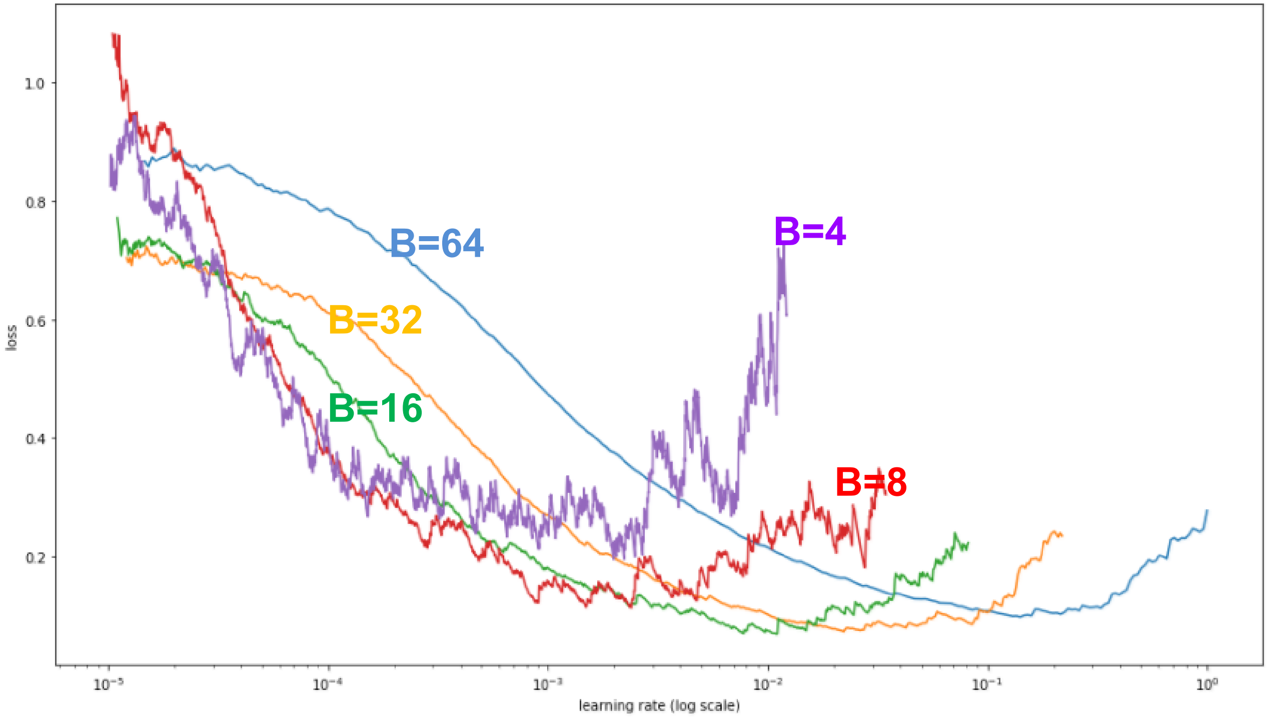
Mini-Batch SGD

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

```
Initialize weights and bias

Random shuffle dataset

BatchSize = 32

for epoch in range(E):

    select #BatchSize samples

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

    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.

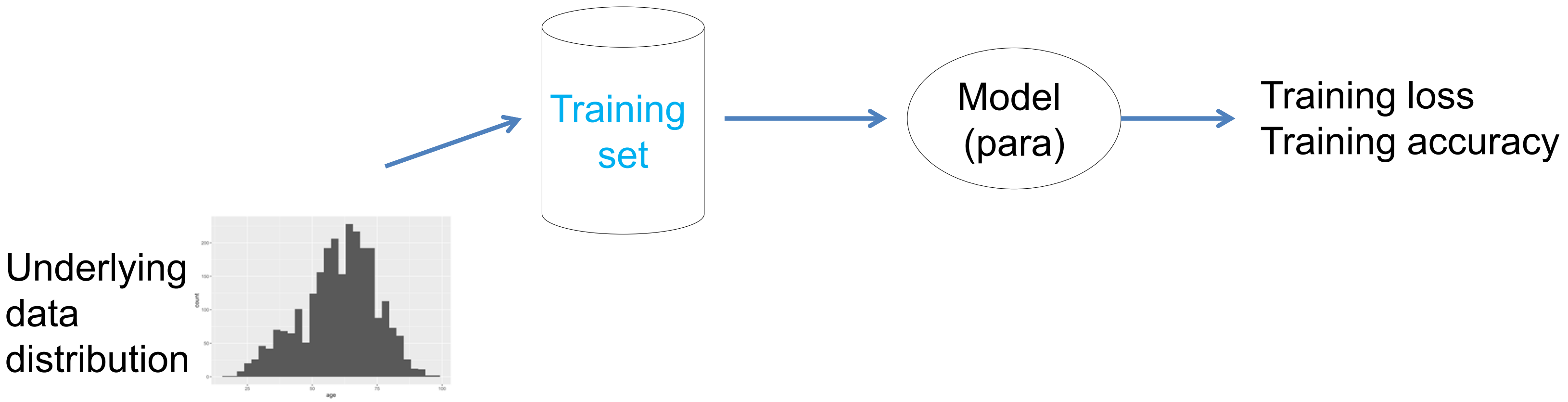
Larger batch size, higher learning rate



- 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

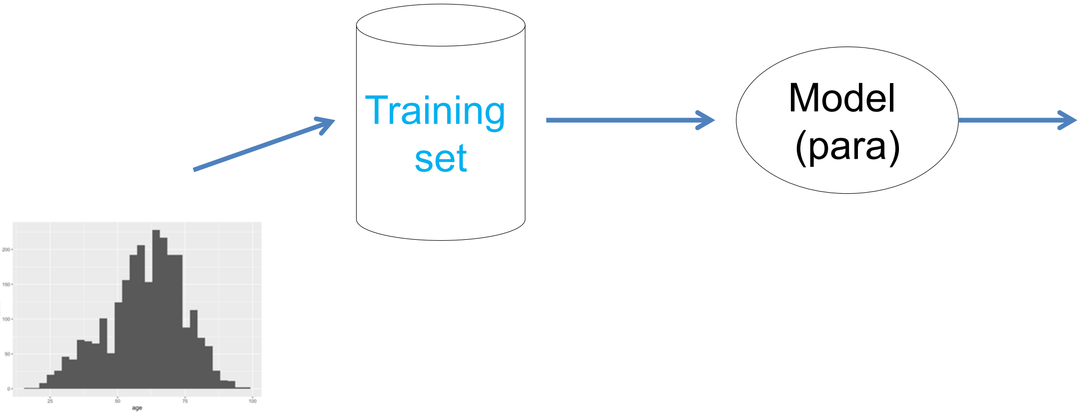


We don’t have this distribution or an effective way to compute it!

- 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

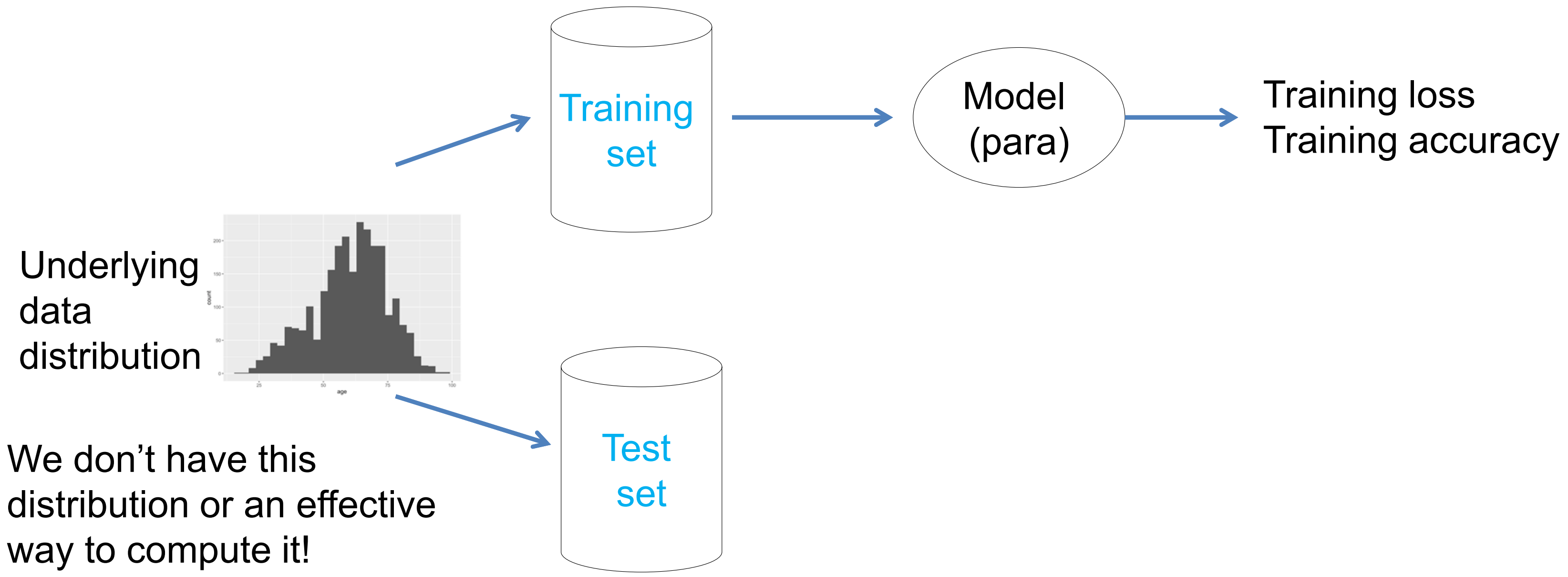


Training loss
Training accuracy

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 set

Model (para)

Training loss
Training accuracy

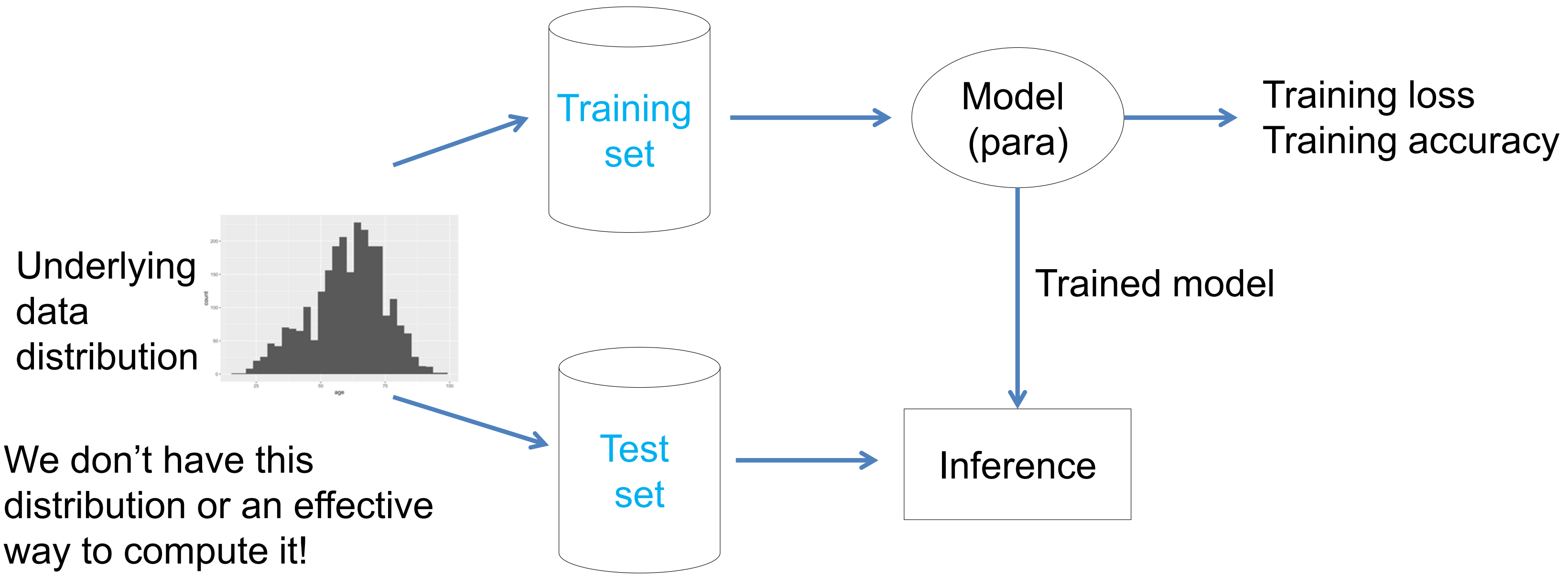
Underlying data distribution

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

Test set

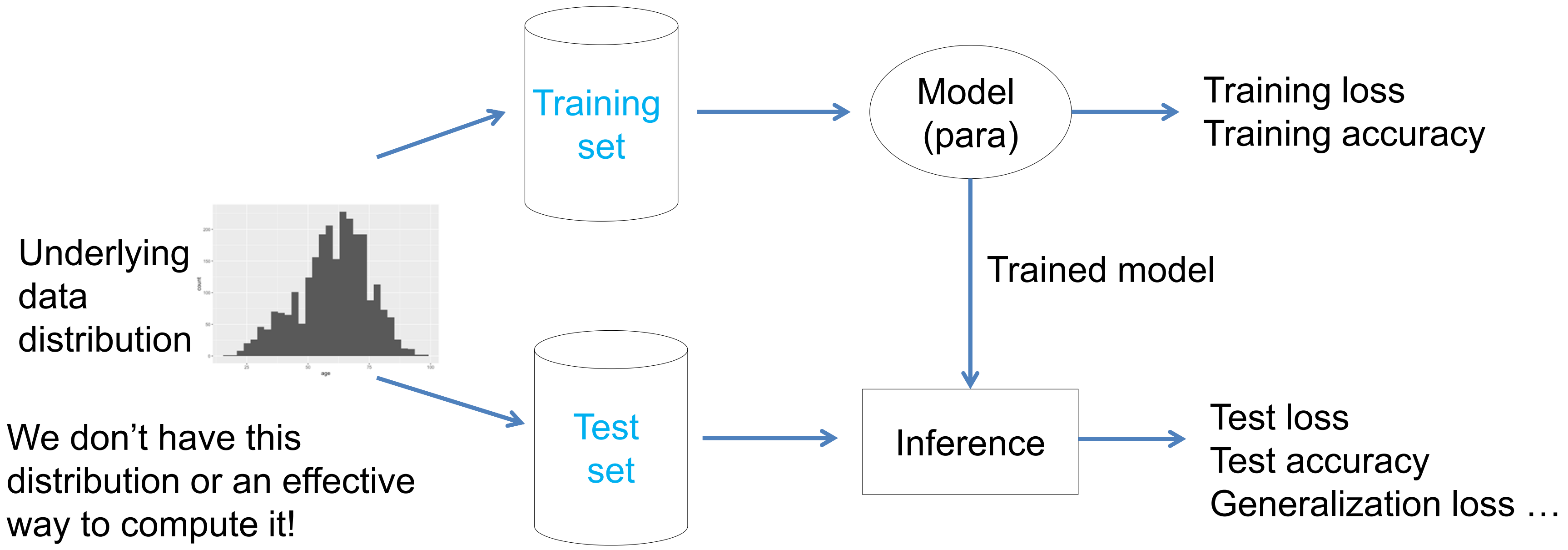
- Training and test sets are sampled from the same distribution, i.i.d. (independent and identically distributed)
- We care test/generalization performance; that is, the performance of training model on a new dataset

Model Generalization


Training set
Model (para)
Training loss
Training accuracy
Underlying data distribution
Trained model
Test set
Inference
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 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:

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

http://statweb.stanford.edu/~tibs/ElemStatLearn/, section 2.9
https://en.wikipedia.org/wiki/Bias%E2%80%93variance_tradeoff

Model error

Model prediction error consists of three parts:

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

$$Bias(\boldsymbol{M},\boldsymbol{D}) = E_{\boldsymbol{D}}[M(\boldsymbol{x};\boldsymbol{D})] - f(\boldsymbol{x})$$

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

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

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

Model error

Model prediction error consists of three parts:

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

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

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

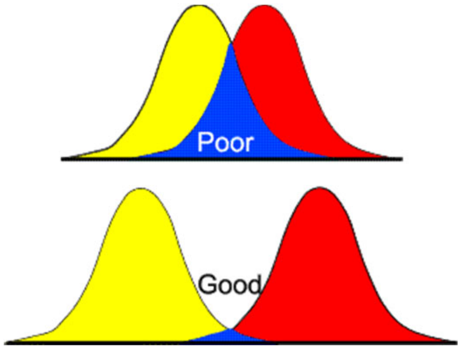
Model prediction error consists of three parts:

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

$\boldsymbol{\sigma}^2$ 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

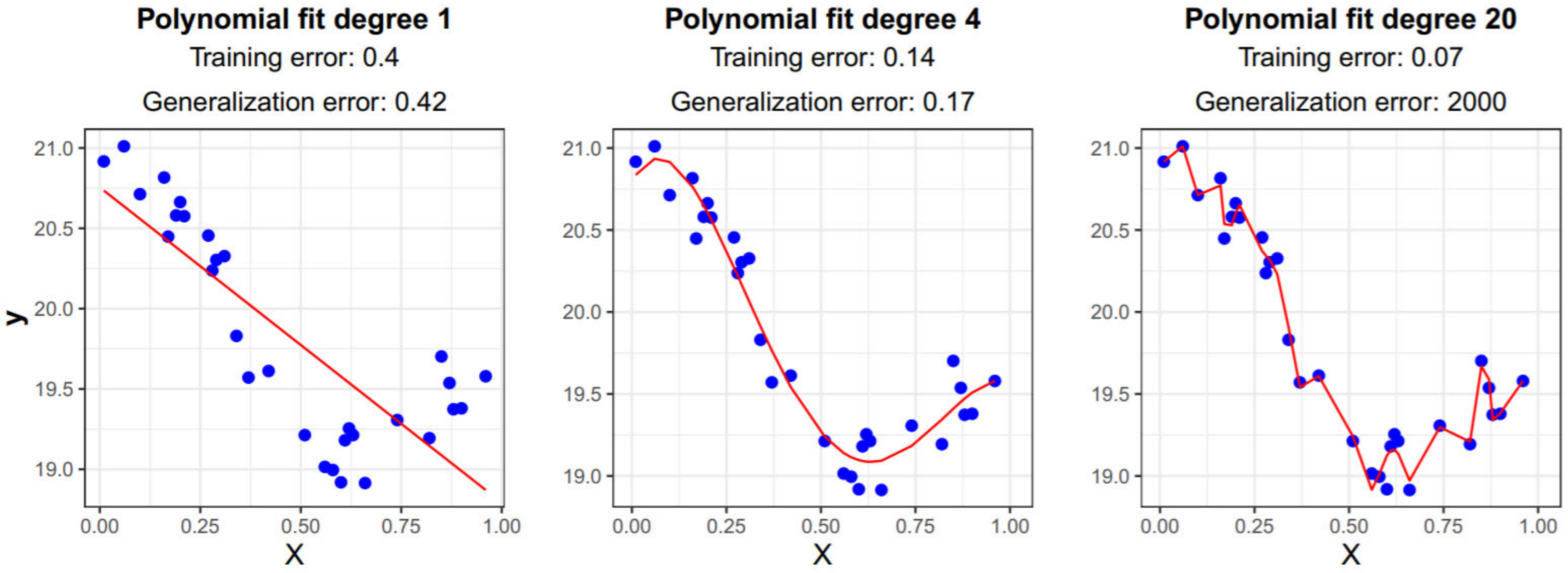


Distribution of two classes can overlap

http://www.alanfielding.co.uk/multivar/dawords.htm

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

Underfitting and overfitting



Underfit

Model cannot represent the data distribution

Good fit

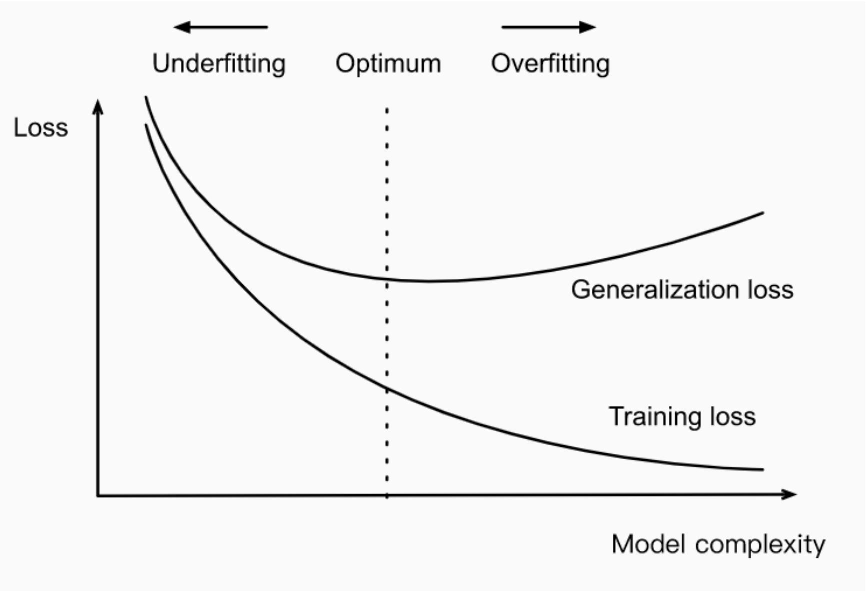
Model well represents the data distribution, and does not capture data noise

Overfit

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

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

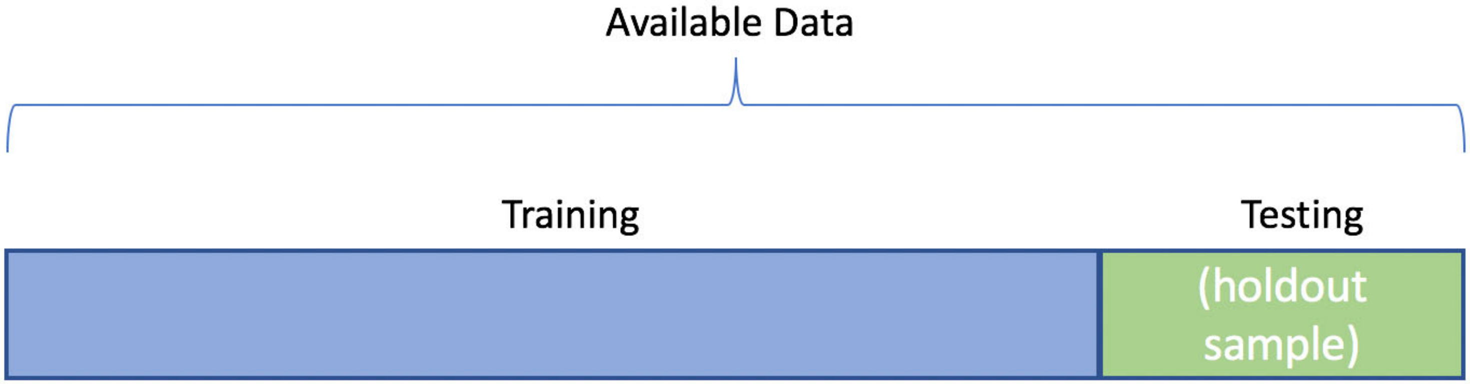
Underfitting and overfitting



- 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



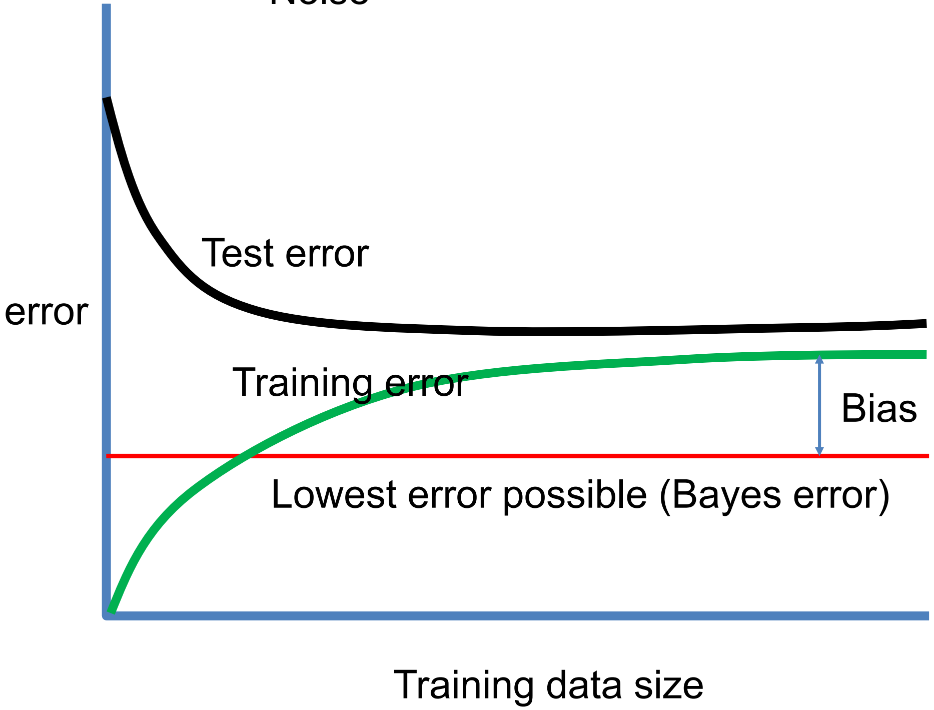
- We train on Training set and use training accuracy to estimate model performance ← **Bias.**

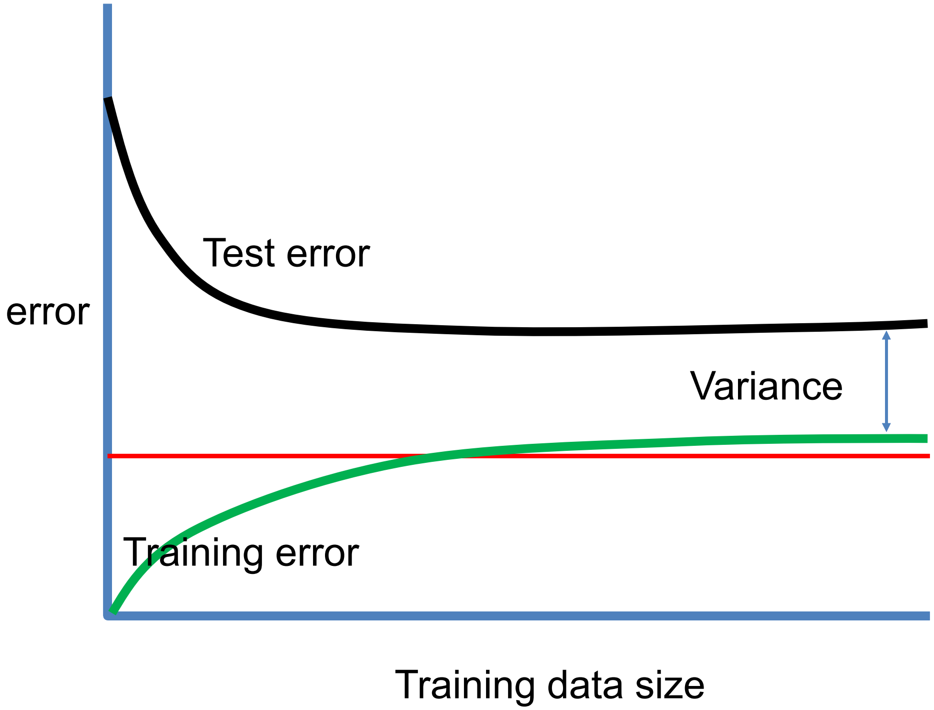
- We apply the trained model on Testing set. Performance difference between training accuracy and test accuracy gives an estimation of **Variance**.

Bias and 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

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

Data loss: how well model fits the data

Regularization loss: prevent model from fitting training data too well

L2 Regularization, weight decay

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

$\|\boldsymbol{W}\|_2^2 = \sum_{k=0}^{P-1} {w_k}^2$ For all parameters in the model, flatten them and computing the element-wise L2 norm

$$\frac{\partial L}{\partial w_k} = \frac{1}{B}\sum_{i=0}^{B-1}\frac{\partial L^{(i)}}{\partial w_k} + \lambda w_k \qquad w_k = w_k - \alpha\left(\frac{1}{B}\sum_{i=0}^{B-1}\frac{\partial L^{(i)}}{\partial w_k}\right) - \underbrace{\alpha\lambda w_k}_{\text{weight decay}}$$

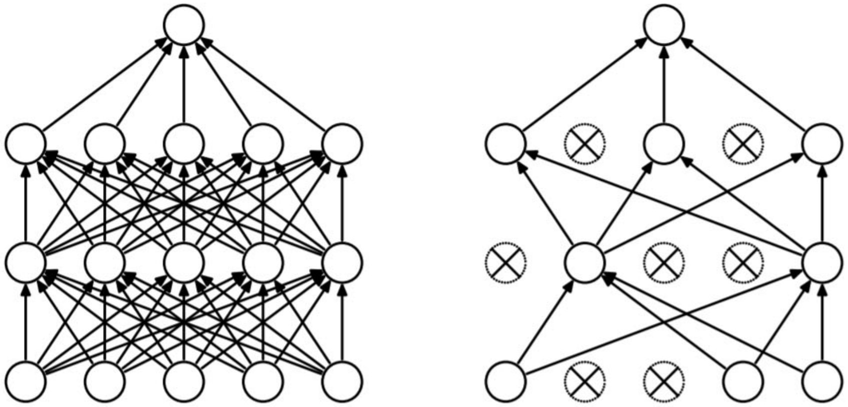
L1 Regularization

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

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

- 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
- Often used with linear layer, not for convolution
- Require more epochs to train

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.

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:

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

…

More on this topic in later lecturers