# Exercise 04 (Learning From Data Caltech)

## I. Student Information


- **Full name:** Lê Phước Phát
- **StudentID:** 22127322
- **Class:** 22KHMT2
- **Email:** lpphat22@clc.fitus.edu.vn


## II. Solving Assignment

### Imported needed libraries

```
# import các thư viện cần thiết
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
from typing import Callable, List, Dict
```

### Generalization Error

In Problems $1-3$, we look at generalization bounds numerically. For $N > d_{\text{vc}}$, use the simple approximate bound $N^{d_{\text{vc}}}$ for the growth function $m_{\mathcal{H}}(N)$.

### Problem $01$

For an $\mathcal{H}$ with $d_{\text{vc}} = 10$, if you want $95\%$ confidence that your generalization error is at most $0.05$, what is the closest numerical approximation of the sample size that the VC generalization bound predicts?

**[a]** $400.000$

**[b]** $420.000$

**[c]** $440.000$

**[d]** $460.000$

**[e]** $480.000$

**Answer**

In this problem $1-3$, firstly, we will prove some formula below:

Case $01$: $N > d_{\text{VC}}$

- Because $d_{\text{VC}} = 10$ is the VC dimension of $\mathcal{H}$, $k = d_{\text{VC}} + 1 = 11$ is a break point for $m_{\mathcal{H}}$. So, $m_{\mathcal{H}}(N) < 2^N$ for all $N > d_{\text{VC}}$. We have

$$m_{\mathcal{H}}(N) \leq \sum_{i=0}^{k-1} \binom{N}{i} = \sum_{i=0}^{d_{\text{VC}}} \binom{N}{i} \approx N^{d_{\text{VC}}} < 2^N \; \forall \; N > d_{\text{VC}} \tag{1.1}$$

- Applying the theorem of generalization bound derived from the VC inequality, in the "good event" with probability of confidence $\geq 1 - \delta > 0$, we have the bound which is

$$|E_{\text{out}} - E_{\text{in}}| \leq \Omega(N, \mathcal{H}, \delta) = \sqrt{\frac{8}{N} \ln(\frac{4m_{\mathcal{H}}(2N)}{\delta})} \tag{1.2}$$

- Now, from $(1)$ and $(2)$, we will have

$$|E_{\text{out}} - E_{\text{in}}| \leq \Omega(N, \mathcal{H}, \delta) = \sqrt{\frac{8}{N}\ln(\frac{4m_{\mathcal{H}}(2N)}{\delta})} \leq \sqrt{\frac{8}{N}\ln(\frac{4\sum_{i=0}^{d_{\text{VC}}}\binom{2N}{i}}{\delta})} \approx \sqrt{\frac{8}{N}\ln(\frac{4(2N)^{d_{\text{VC}}}}{\delta})}\ \forall\ N > d_{\text{VC}} \quad (1.3)$$

- Because we want the generalization error to be at most $\epsilon$, which means that $\Omega(N, \mathcal{H}, \delta) \leq \epsilon$, so from $(3)$ we have

$$\sqrt{\frac{8}{N}\ln(\frac{4(2N)^{d_{\text{VC}}}}{\delta})} \leq \epsilon \Leftrightarrow N \geq \frac{8}{\epsilon^2}\ln(\frac{4(2N)^{d_{\text{VC}}}}{\delta})\ \forall\ N > d_{\text{VC}} \quad (1.4)$$

Case $02$: $N \leq d_{\text{VC}}$

- Beside that, if $N \leq d_{\text{VC}}$, then $m_{\mathcal{H}}$ has no break point so

$$m_{\mathcal{H}}(N) = 2^N\ \forall\ N \leq d_{\text{VC}} \quad (1.5)$$

- From $(2)$ and $(5)$, we will have

$$|E_{\text{out}} - E_{\text{in}}| \leq \Omega(N, \mathcal{H}, \delta) = \sqrt{\frac{8}{N}\ln(\frac{4m_{\mathcal{H}}(2N)}{\delta})} = \sqrt{\frac{8}{N}\ln(\frac{4 \times 2^{2N}}{\delta})}\ \forall\ N \leq d_{\text{VC}} \quad (1.6)$$

- For this case $N \leq d_{\text{VC}}$, we also want the generalization error to be at most $\epsilon$, so from $(6)$ we have

$$\sqrt{\frac{8}{N}\ln(\frac{4 \times 2^{2N}}{\delta})} \leq \epsilon \Leftrightarrow N \geq \frac{8}{\epsilon^2}\ln(\frac{4 \times 2^{2N}}{\delta})\ \forall\ N \leq d_{\text{VC}} \quad (1.7)$$

In this problem $1$ , we have:

- $1 - \delta = 95\% = 0.95 \implies \delta = 0.05$
- $d_{\text{VC}} = 10$
- $\epsilon = 0.05$

Now, we will solve this inequality $(1.4)$ and $(1.7)$ by some code below, and find the minimum numerical approximation of the sample size $N_{\text{min}}$ that the VC generalization bound predicts.

```
# Khai báo các tham số cần thiết
delta = 0.05 # xác suất lỗi tin cậy
d_vc = 10 # VC dimension
epsilon = 0.05 # sai số tổng quát mong muốn

print("All parameters are initialized:")
print(f"delta = {delta}, d_vc = {d_vc}, epsilon = {epsilon}")
```

```
All parameters are initialized:
delta = 0.05, d_vc = 10, epsilon = 0.05
```

```
# Functions to solve inequalities
# Case 01: N > d_vc
def inequality_case_01(N):
  if N <= d_vc:
    return float('inf') # exclude invalid N
  term1 = 8 / epsilon**2
  term2 = np.log((4 * (2 * N)**d_vc) / delta)
  return term1 * term2 - N

# Case 02: N <= d_vc
def inequality_case_02(N):
  if N > d_vc:
    return float('inf') # exclude invalid N
  term1 = 8 / epsilon**2
  term2 = np.log((4 * 2**(2 * N)) / delta)
  return term1 * term2 - N

# Find the minimum number sample size
def find_numerical_sample_size():
  N_min_case_1 = None
  N_min_case_2 = None

  # Case 01 N > d_vc
  for N in range(d_vc + 1, int(1e6)):
    if inequality_case_01(N) <= 0:
      N_min_case_1 = N
      break
    else:
      N_min_case_1 = float('inf')

```

```
  # Case 02:
  for N in range(1, d_vc + 1):
    if inequality_case_02(N) <= 0:
      N_min_case_2 = N
      break
    else:
      N_min_case_2 = float('inf')

  return min(N_min_case_1, N_min_case_2)

```

```
def find_nearest_answer(result, choices):
  nearest_answer = min(choices, key = lambda x: abs(x - result))
  return nearest_answer, result
```

```
choices = [400000, 420000, 440000, 460000, 480000]
N_min = find_numerical_sample_size()
nearest_answer, result = find_nearest_answer(N_min, choices)
print(f"The nearest answer among the given choices is {nearest_answer} with the final result {result}.")
```

```
The nearest answer among the given choices is 460000 with the final result 452957.
```

Finally, the correct answer is **[d]** $460.000$

### Problem $02$

There are a number of bounds on the generalization error $\epsilon$, all holding with probability at least $1-\delta$. Fix $d_{\mathrm{vc}}=50$ and $\delta=0.05$ and plot these bounds as a function of $N$. Which bound is the smallest for very large $N$, say $N=10.000$ ? Note that **[c]** and **[d]** are implicit bounds in $\epsilon$.

**[a]** Original VC bound: $\epsilon \leq \sqrt{\frac{8}{N}\ln(\frac{4m_{\mathcal{H}}(2N)}{\delta})}$

**[b]** Rademacher Penalty Bound: $\epsilon \leq \sqrt{\frac{2\ln(2Nm_{\mathcal{H}}(N))}{N}}+\sqrt{\frac{2}{N}\ln\frac{1}{\delta}}+\frac{1}{N}$

**[c]** Parrondo and Van den Broek: $\epsilon \leq \sqrt{\frac{1}{N}(2\epsilon+\ln(\frac{6m_{\mathcal{H}}(2N)}{\delta}))}$

**[d]** Devroye: $\epsilon \leq \sqrt{\frac{1}{2N}(4\epsilon(1+\epsilon)+\ln\frac{4m_{\mathcal{H}}(N^2)}{\delta})}$

**[e]** They are all equal.

**Answer**

In this problem $02$, because $N=10000>d_{\mathrm{VC}}=50$ (case $01$), and $\delta=0.05$, we will apply inequality $(1.1)$ for these above bounds as a function of $N$.

```
# khai báo các tham số cần thiết
N_target = 10000
d_vc = 50
delta = 0.05

print("All parameters are initialized:")
print(f"N = {N_target}, d_vc = {d_vc}, delta = {delta}")
```

```
All parameters are initialized:
N = 10000, d_vc = 50, delta = 0.05
```

Now, we will preprocess these bounds to plot several different bounds on the generalization error $\epsilon$, all holding with probability at least $1-\delta$.

**[a]** Original VC bound

$$
\begin{aligned}
\epsilon &\leq \sqrt{\frac{8}{N}\ln(\frac{4m_{\mathcal{H}}(2N)}{\delta})} \\
&\leq \sqrt{\frac{8}{N}\ln(\frac{4\sum_{i=0}^{d_{\mathrm{VC}}}\binom{2N}{i}}{\delta})} \\
&\approx \sqrt{\frac{8}{N}\ln(\frac{4(2N)^{d_{\mathrm{VC}}}}{\delta})} \\
&= \sqrt{\frac{8}{N}((d_{\mathrm{VC}}+2)\times\ln(2)+d_{\mathrm{VC}}\times\ln(N)-\ln(\delta))}\ \forall\ N > d_{\mathrm{VC}}
\end{aligned} \tag{2.1}
$$

```
def original_vc_bound(N, d_vc = d_vc, delta = delta):
  try:
    term1 = 8 / N
    term2 = (d_vc + 2) * np.log(2)
    term3 = d_vc * np.log(N)
    term4 = np.log(delta)
    bound = np.sqrt(term1 * (term2 + term3 - term4))
    return bound
  except Exception as e:
    print(f"Error in original_vc_bound: {e}")
    return None
```

```
print(f"Original VC bound at N = {N_target} is: {original_vc_bound(N_target, d_vc, delta)}")
```

```
Original VC bound at N = 10000 is: 0.632174915200836
```

**[b]** Rademacher Penalty Bound

$$
\begin{aligned}
\epsilon &\leq \sqrt{\frac{2\ln(2Nm_{\mathcal{H}}(N))}{N}} + \sqrt{\frac{2}{N}\ln\frac{1}{\delta}} + \frac{1}{N} \\
&\leq \sqrt{\frac{2\ln(2N\times\sum_{i=0}^{d_{\mathrm{VC}}}\binom{N}{i})}{N}} + \sqrt{\frac{2}{N}\ln\frac{1}{\delta}} + \frac{1}{N} \\
&\approx \sqrt{\frac{2\ln(2(N)^{d_{\mathrm{VC}}+1})}{N}} + \sqrt{\frac{2}{N}\ln\frac{1}{\delta}} + \frac{1}{N} \\
&= \sqrt{\frac{2\ln(2)+2(d_{\mathrm{VC}}+1)\ln(N)}{N}} + \sqrt{\frac{2}{N}\ln\frac{1}{\delta}} + \frac{1}{N}\ \forall\ N > d_{\mathrm{VC}}
\end{aligned} \tag{2.2}
$$

```
def rademacher_penalty_bound(N, d_vc=50, delta=0.05):
  try:
    term1 = np.sqrt(((2 * np.log(2)) + 2 * (d_vc + 1) * np.log(N)) / N)
    term2 = np.sqrt((2 / N) * np.log(1 / delta))
    term3 = 1 / N
    bound = term1 + term2 + term3
    return bound
  except Exception as e:
    print(f"Error in rademacher_penalty_bound: {e}")
    return None
```

```
print(f"Rademacher Penalty bound at N = {N_target} is: {rademacher_penalty_bound(N_target, d_vc, delta)}")
```

```
Rademacher Penalty bound at N = 10000 is: 0.3313087859616395
```

**[c]** Parrondo and Van den Broek

$$
\begin{aligned}
\epsilon &\leq \sqrt{\frac{1}{N}(2\epsilon + \ln(\frac{6m_{\mathcal{H}}(2N)}{\delta}))} \\
&\leq \sqrt{\frac{1}{N}(2\epsilon + \ln(\frac{6\sum_{i=0}^{d_{\mathrm{VC}}}\binom{2N}{i}}{\delta})} \\
&\approx \sqrt{\frac{1}{N}(2\epsilon + \ln(\frac{6(2N)^{d_{\mathrm{VC}}}}{\delta}))} \\
&= \sqrt{\frac{1}{N}(2\epsilon + \ln(6) + d_{\mathrm{VC}} \times \ln(2) + d_{\mathrm{VC}} \times \ln(N) - \ln(\delta))} \; \forall \; N > d_{\mathrm{VC}}
\end{aligned} \tag{2.3}
$$

```python
def parrondo_van_den_broek_bound(N, d_vc=d_vc, delta=delta, epsilon_init=0.1, tol=1e-6, max_iter=1000):
    try:
      epsilon = epsilon_init
      for _ in range(max_iter):
          # tính các giá trị logarit
          term1 = 1 / N
          term2 = 2 * epsilon
          term3 = np.log(6) + d_vc * np.log(2) + d_vc * np.log(N) - np.log(delta)

          # tính epsilon mới
          new_epsilon = np.sqrt(term1 * (term2 + term3))

          # kiểm tra điều kiện hội tụ
          if abs(new_epsilon - epsilon) < tol:
              return new_epsilon

          # cập nhât epsilon mới cho vòng lặp tiếp theo
          epsilon = new_epsilon

      # Nếu vượt quá số vòng lặp tối đa mà không hội tụ
      raise ValueError("Parrondo and Van den Broek bound did not converge.")

    except Exception as e:
      print(f"Error in parrondo_van_den_broek_bound: {e}")
      return None
```

```python
print(f"Parrondo and Van den Broek bound at N = {N_target} is: {parrondo_van_den_broek_bound(N_target, d_vc, delta)}")
```

```
Parrondo and Van den Broek bound at N = 10000 is: 0.2236982936697339
```

**[d]** Devroye

$$
\begin{aligned}
\epsilon &\leq \sqrt{\frac{1}{2N}(4\epsilon(1+\epsilon) + \ln(\frac{4m_{\mathcal{H}}(N^2)}{\delta}))} \\
&\leq \sqrt{\frac{1}{2N}(4\epsilon(1+\epsilon) + \ln(\frac{4\sum_{i=0}^{d_{\mathrm{VC}}}\binom{N^2}{i}}{\delta}))} \\
&\approx \sqrt{\frac{1}{2N}(4\epsilon(1+\epsilon) + \ln(\frac{4(N)^{2d_{\mathrm{VC}}}}{\delta}))} \\
&= \sqrt{\frac{1}{2N}(4\epsilon(1+\epsilon) + \ln(4) + 2d_{\mathrm{VC}}\ln(N) - \ln(\delta))} \; \forall \; N > d_{\mathrm{VC}}
\end{aligned} \tag{2.4}
$$

```python
# Hàm tính Devroye bound
def devroye_bound(N, d_vc=d_vc, delta=delta, epsilon_init=0.1, tol=1e-6, max_iter=1000):
    try:
      epsilon = epsilon_init  # Giá trị khởi tạo của epsilon
      for iteration in range(max_iter):
          # Tính các giá trị logarit
          term1 = 1 / (2 * N)
          term2 = 4 * epsilon * (1 + epsilon)
          term3 = np.log(4) + 2 * d_vc * np.log(N) - np.log(delta)

          # Tính epsilon mới
```

```
          new_epsilon = np.sqrt(term1 * (term2 + term3))

          # Kiểm tra điều kiện hội tụ
          if abs(new_epsilon - epsilon) < tol:
              return new_epsilon

          # Cập nhật epsilon cho vòng lặp tiếp theo
          epsilon = new_epsilon

      # Nếu không hội tụ sau max_iter vòng lặp, báo lỗi
      raise ValueError("Devroye bound did not converge.")

    except Exception as e:
      print(f"Error in devroye_bound: {e}")
      return None
```

```
print(f"Devroye bound at N = {N_target} is: {devroye_bound(N_target, d_vc, delta)}")
```

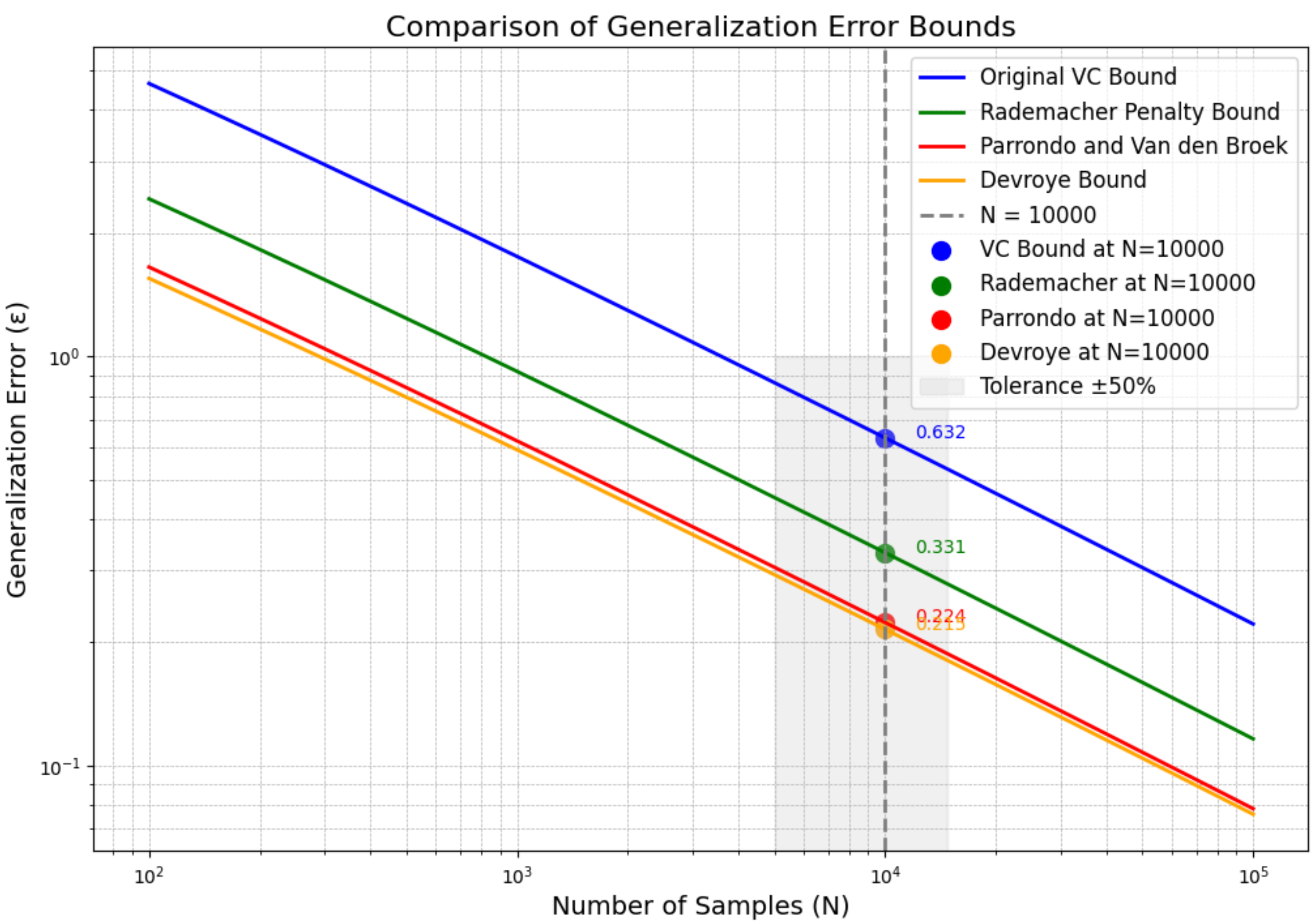
```
Devroye bound at N = 10000 is: 0.21522804977713944
```

Now, we will plot all above preprocessed bounds to visualize all above bounds for finding the smallest generalization error bound.

```
# hàm vẽ đồ thị các generalization error bounds
def plot_bounds(N_values, d_vc = d_vc, delta = delta, N_target = None, tolerance = 0.5):
  """
  Hàm vẽ đồ thị các bounds
  """

  try:
    # tính tất cả các bounds cho tập N_values
    vc_bounds = [original_vc_bound(N) for N in N_values]
    rademacher_penalty_bounds = [rademacher_penalty_bound(N) for N in N_values]
    parrondo_van_den_broek_bounds = [parrondo_van_den_broek_bound(N) for N in N_values]
    devroye_bounds = [devroye_bound(N) for N in N_values]

    # Vẽ biểu đồ
    plt.figure(figsize=(12, 8))
    # Vẽ các đường bounds
    plt.plot(N_values, vc_bounds, label="Original VC Bound", color="blue", linewidth=2)
    plt.plot(N_values, rademacher_penalty_bounds, label="Rademacher Penalty Bound", color="green", linewidth=2)
    plt.plot(N_values, parrondo_van_den_broek_bounds, label="Parrondo and Van den Broek", color="red", linewidth=2)
    plt.plot(N_values, devroye_bounds, label="Devroye Bound", color="orange", linewidth=2)

    # Nếu có target, vẽ các điểm giao
    if N_target:
      # tính các giá trị của từng bound tại N_target
      vc_at_target = original_vc_bound(N_target)
      rademacher_at_target = rademacher_penalty_bound(N_target)
      parrondo_at_target = parrondo_van_den_broek_bound(N_target)
      devroye_at_target = devroye_bound(N_target)

      # vẽ đường thẳng đứng tại N_target
      plt.axvline(x=N_target, color='gray', linestyle='--', linewidth=2, label=f"N = {N_target}")

      # tô các tọa độ điểm tại N_target
      plt.scatter([N_target], [vc_at_target], color="blue", s=100, label=f"VC Bound at N={N_target}")
      plt.scatter([N_target], [rademacher_at_target], color="green", s=100, label=f"Rademacher at N={N_target}")
      plt.scatter([N_target], [parrondo_at_target], color="red", s=100, label=f"Parrondo at N={N_target}")
      plt.scatter([N_target], [devroye_at_target], color="orange", s=100, label=f"Devroye at N={N_target}")

      # điền các giá tri số tại các điểm giao
      plt.text(N_target * 1.2, vc_at_target, f"{vc_at_target:.3f}", fontsize=10, color="blue")
      plt.text(N_target * 1.2, rademacher_at_target, f"{rademacher_at_target:.3f}", fontsize=10, color="green")
      plt.text(N_target * 1.2, parrondo_at_target, f"{parrondo_at_target:.3f}", fontsize=10, color="red")
      plt.text(N_target * 1.2, devroye_at_target, f"{devroye_at_target:.3f}", fontsize=10, color="orange")

      # tô bóng vùng xung quanh N_target với khoảng tolerance
      lower_bound = N_target * (1 - tolerance)
      upper_bound = N_target * (1 + tolerance)
      plt.fill_between(N_values, 0, 1, where=(N_values > lower_bound) & (N_values < upper_bound), color='lightgray', alpha=0.3, label=f"T

    # Định dạng biểu đồ
    plt.xscale("log")
    plt.yscale("log")
    plt.xlabel("Number of Samples (N)", fontsize=14)
```

```
    plt.ylabel("Generalization Error (ε)", fontsize=14)
    plt.title("Comparison of Generalization Error Bounds", fontsize=16)
    plt.legend(fontsize=12, loc='upper right')
    plt.grid(True, which="both", linestyle="--", linewidth=0.5)

    # Hiển thị biểu đồ
    plt.show()

  except Exception as e:
    print(f"Error in plot_bounds: {e}")
    return None
```

```
N_values = np.logspace(2, 5, 500) # tập N gồm 500 điểm khác nhau từ 100 đến 100000
plot_bounds(N_values, d_vc = d_vc, delta = delta, N_target = N_target)
```




Now, we will find the smallest for very large $N$, say $N = 10.000$.

```
# hàm tìm smallest VC Bound tại N_target
def find_smallest_VC_bound(N_target, d_vc = d_vc, delta = delta):
  try:
    # tính các giá trị bound tại N_target
    vc_bound_value = original_vc_bound(N_target, d_vc, delta)
    rademacher_penalty_bound_value = rademacher_penalty_bound(N_target, d_vc, delta)
    parrondo_van_den_broek_bound_value = parrondo_van_den_broek_bound(N_target, d_vc, delta)
    devroye_bound_value = devroye_bound(N_target, d_vc, delta)

    # lưu trữ các giá trị bounds với tên tương ứng
    bounds = {
        "Original VC Bound": vc_bound_value,
        "Rademacher Penalty Bound": rademacher_penalty_bound_value,
        "Parrondo and Van den Broek Bound": parrondo_van_den_broek_bound_value,
        "Devroye Bound": devroye_bound_value
    }

    # tìm giá trị bound nhỏ nhất
    smallest_bound_name = min(bounds, key=bounds.get)
```

```
    smallest_bound_value = bounds[smallest_bound_name]

    # trả về kết quả
    return smallest_bound_name, smallest_bound_value

  except Exception as e:
    print(f"Error in find_smallest_VC_bound: {e}")
    return None
```

```
smallest_bound_name, smallest_bound_value = find_smallest_VC_bound(N_target, d_vc, delta)
print(f"The smallest VC Bound is {smallest_bound_name} with value {smallest_bound_value:.6f}.")
```

```
The smallest VC Bound is Devroye Bound with value 0.215228.
```

Finally, the correct answer is **[d]** Devroye: $\epsilon \leq \sqrt{\frac{1}{2N}(4\epsilon(1+\epsilon)+\ln\frac{4m_{\mathcal{H}}(N^2)}{\delta})}$

## Problem $03$

For the same values of $d_{\mathrm{VC}}$ and $\delta$ of Problem $2$, but for small $N$, say $N=5$, which bound is the smallest?

**[a]** Original VC bound: $\epsilon \leq \sqrt{\frac{8}{N}\ln(\frac{4m_{\mathcal{H}}(2N)}{\delta})}$

**[b]** Rademacher Penalty Bound: $\epsilon \leq \sqrt{\frac{2\ln(2Nm_{\mathcal{H}}(N))}{N}} + \sqrt{\frac{2}{N}\ln\frac{1}{\delta}} + \frac{1}{N}$

**[c]** Parrondo and Van den Broek: $\epsilon \leq \sqrt{\frac{1}{N}(2\epsilon + \ln(\frac{6m_{\mathcal{H}}(2N)}{\delta}))}$

**[d]** Devroye: $\epsilon \leq \sqrt{\frac{1}{2N}(4\epsilon(1+\epsilon)+\ln(\frac{4m_{\mathcal{H}}(N^2)}{\delta}))}$

**[e]** They are all equal.

**Answer**

In this problem $03$, we have $d_{\mathrm{VC}} = 50 > N = 5$ (case $02$), and $\delta = 0.05$, so we will apply the formular $(1.5)$ for these above bounds as a function of $N$.

```
# khai báo các tham số cần thiết
N_target = 5
d_vc = 50
delta = 0.05

print("All parameters are initialized:")
print(f"N = {N_target}, d_vc = {d_vc}, delta = {delta}")
```

```
All parameters are initialized:
N = 5, d_vc = 50, delta = 0.05
```

Now, we will preprocess these bounds to plot several different bounds on the generalization error $\epsilon$, all holding with probability at least $1-\delta$.

**[a]** Original VC bound

$$\epsilon \leq \sqrt{\frac{8}{N}\ln(\frac{4m_{\mathcal{H}}(2N)}{\delta})} = \sqrt{\frac{8}{N}\ln(\frac{4\times 2^{2N}}{\delta})} = \sqrt{\frac{8}{N}((2N+2)\times\ln(2)-\ln(\delta))}\ \forall\, N \leq d_{\mathrm{VC}} \tag{3.1}$$

So, we will update the function `original_vc_bound()` to implement both of $2$ cases ($N > d_{\mathrm{VC}}$ and $N \leq d_{\mathrm{VC}}$).

```
def original_vc_bound(N, d_vc = d_vc, delta = delta):
  try:
    #case 01: N > d_vc
    if N > d_vc:
      term1 = 8 / N
      term2 = (d_vc + 2) * np.log(2)
      term3 = d_vc * np.log(N)
      term4 = np.log(delta)
```

```
      bound = np.sqrt(term1 * (term2 + term3 - term4))
    # case 02: N <= d_vc
    elif N <= d_vc:
      term1 = 8 / N
      term2 = (2 * N + 2) * np.log(2) - np.log(delta)
      bound = np.sqrt(term1 * term2)

    return bound
  except Exception as e:
    print(f"Error in original_vc_bound: {e}")
    return None
```

```
print(f"Original VC bound at N = {N_target} is: {original_vc_bound(N_target, d_vc, delta)}")
```

```
Original VC bound at N = 5 is: 4.254597220000659
```

**[b]** Rademacher Penalty Bound

$$\begin{aligned}\epsilon &\leq \sqrt{\frac{2\ln(2Nm_{\mathcal{H}}(N))}{N}} + \sqrt{\frac{2}{N}\ln\frac{1}{\delta}} + \frac{1}{N} \\ &= \sqrt{\frac{2\ln(2N \times 2^N)}{N}} + \sqrt{\frac{2}{N}\ln\frac{1}{\delta}} + \frac{1}{N} \\ &= \sqrt{\frac{2(\ln(N) + (N+1)\times\ln(2))}{N}} + \sqrt{\frac{2}{N}\ln\frac{1}{\delta}} + \frac{1}{N}\ \forall\ N \leq d_{\mathrm{VC}} \end{aligned} \tag{3.2}$$

So, we will update the function `rademacher_penalty_bound()` to implement both of 2 cases ($N > d_{\mathrm{VC}}$ and $N \leq d_{\mathrm{VC}}$).

```
def rademacher_penalty_bound(N, d_vc = d_vc, delta = delta):
  try:
    # case 01: N > d_vc
    if N > d_vc:
      term1 = np.sqrt(((2 * np.log(2)) + 2 * (d_vc + 1) * np.log(N)) / N)
      term2 = np.sqrt((2 / N) * np.log(1 / delta))
      term3 = 1 / N
      bound = term1 + term2 + term3
    # case 02: N <= d_vc
    elif N <= d_vc:
      term1 = np.sqrt((2 * (np.log(N) + (N + 1) * np.log(2))) / N)
      term2 = np.sqrt((2 / N) * np.log(1 / delta))
      term3 = 1 / N
      bound = term1 + term2 + term3
    return bound
  except Exception as e:
    print(f"Error in rademacher_penalty_bound: {e}")
    return None
```

```
print(f"Rademacher Penalty bound at N = {N_target} is: {rademacher_penalty_bound(N_target, d_vc, delta)}")
```

```
Rademacher Penalty bound at N = 5 is: 2.813654929686762
```

**[c]** Parrondo and Van den Broek

$$\begin{aligned}\epsilon &\leq \sqrt{\frac{1}{N}(2\epsilon + \ln(\frac{6m_{\mathcal{H}}(2N)}{\delta}))} \\ &= \sqrt{\frac{1}{N}(2\epsilon + \ln(\frac{6\times 2^{2N}}{\delta}))} \\ &= \sqrt{\frac{1}{N}(2\epsilon + \ln(6) + (2N)\times\ln(2) - \ln(\delta))}\ \forall\ N \leq d_{\mathrm{VC}} \end{aligned} \tag{3.3}$$

So, we will update the function `parrondo_van_den_broek_bound()` to implement both of 2 cases ($N > d_{\mathrm{VC}}$ and $N \leq d_{\mathrm{VC}}$).

```
def parrondo_van_den_broek_bound(N, d_vc = d_vc, delta = delta, epsilon_init = 0.1, tol = 1e-6, max_iter = 1000):
  try:
    epsilon = epsilon_init
    for _ in range(max_iter):

      # case 01: N > d_vc
      if N > d_vc:
        # tính các giá trị logarit
```

```
        term1 = 1 / N
        term2 = 2 * epsilon
        term3 = np.log(6) + d_vc * np.log(2) + d_vc * np.log(N) - np.log(delta)
        # tính epsilon mới
        new_epsilon = np.sqrt(term1 * (term2 + term3))

      # case 02: N <= d_vc
      elif N <= d_vc:
        # tính các giá trị logarit
        term1 = 1 / N
        term2 = 2 * epsilon
        term3 = np.log(6) + (2 * N) * np.log(2) - np.log(delta)
        # tính epsilon mới
        new_epsilon = np.sqrt(term1 * (term2 + term3))

      # kiểm tra điều kiện hội tụ
      if abs(new_epsilon - epsilon) < tol:
        return new_epsilon

      # cập nhât epsilon mới cho vòng lặp tiếp theo
      epsilon = new_epsilon
    # Nếu vượt quá số vòng lặp tối đa mà không hội tụ
    raise ValueError("Parrondo and Van den Broek bound did not converge.")
  except Exception as e:
    print(f"Error in parrondo_van_den_broek_bound: {e}")
    return None
```

```
print(f"Parrondo and Van den Broek bound at N = {N_target} is: {parrondo_van_den_broek_bound(N_target, d_vc, delta)}")
```

```
Parrondo and Van den Broek bound at N = 5 is: 1.7439535444240137
```

**[d]** Devroye

$$\begin{aligned}\epsilon &\leq \sqrt{\frac{1}{2N}(4\epsilon(1+\epsilon)+\ln(\frac{4m_{\mathcal{H}}(N^2)}{\delta}))} \\ &= \sqrt{\frac{1}{2N}(4\epsilon(1+\epsilon)+\ln(\frac{4\times 2^{N^2}}{\delta}))} \\ &= \sqrt{\frac{1}{2N}(4\epsilon(1+\epsilon)+(N^2+2)\times\ln(2)-\ln(\delta))}\ \forall\ N \leq d_{\mathrm{VC}}\end{aligned} \tag{3.4}$$

So, we will update the function `devroye_bound()` to implement both of $2$ cases ($N > d_{\mathrm{VC}}$ and $N \leq d_{\mathrm{VC}}$).

```
# Hàm tính Devroye bound
def devroye_bound(N, d_vc=d_vc, delta=delta, epsilon_init=0.1, tol=1e-6, max_iter=1000):
  try:
    epsilon = epsilon_init  # Giá trị khởi tạo của epsilon
    for iteration in range(max_iter):

      # case 01: N > d_vc
      if N > d_vc:

        # Tính các giá trị logarit
        term1 = 1 / (2 * N)
        term2 = 4 * epsilon * (1 + epsilon)
        term3 = np.log(4) + 2 * d_vc * np.log(N) - np.log(delta)

        # Tính epsilon mới
        new_epsilon = np.sqrt(term1 * (term2 + term3))

      # case 02: N <= d_vc
      elif N <= d_vc:

        # tính các giá trị logarit
        term1 = 1 / (2 * N)
        term2 = 4 * epsilon * (1 + epsilon)
        term3 = (N ** 2 + 2) * np.log(2) - np.log(delta)

        # tính epsilon mới
        new_epsilon = np.sqrt(term1 * (term2 + term3))

      # Kiểm tra điều kiện hội tụ
      if abs(new_epsilon - epsilon) < tol:
          return new_epsilon
```

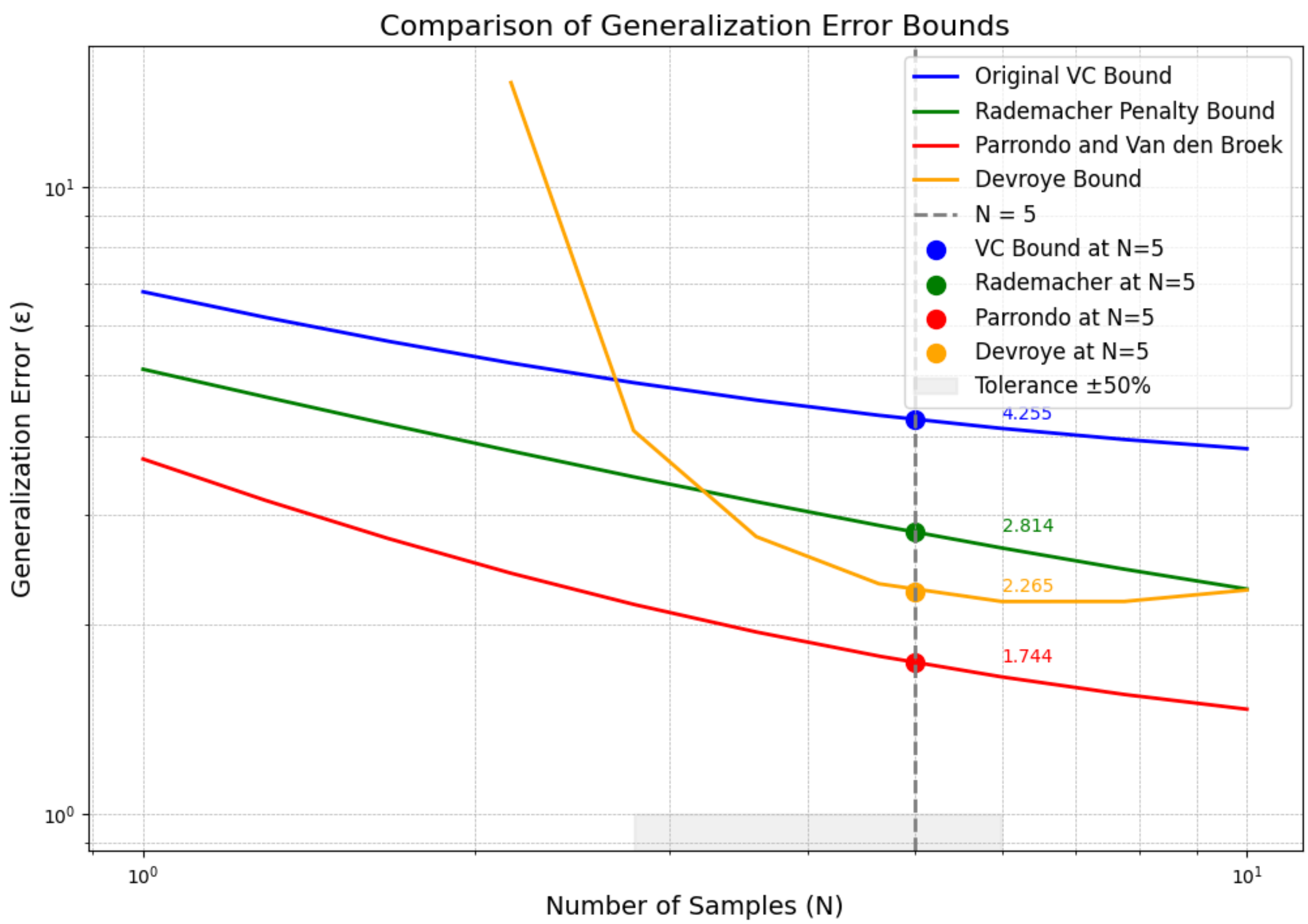
```

        # Cập nhật epsilon cho vòng lặp tiếp theo
        epsilon = new_epsilon

      # Nếu không hội tụ sau max_iter vòng lặp, báo lỗi
      raise ValueError("Devroye bound did not converge.")

    except Exception as e:
      # print(f"Error in devroye_bound: {e}")
      return np.nan
```

```
print(f"Devroye bound at N = {N_target} is: {devroye_bound(N_target, d_vc, delta)}")
```

```
Devroye bound at N = 5 is: 2.2645399272361755
```

Now, we will plot all above preprocessed bounds to visualize all above bounds for finding the smallest generalization error bound.

```
N_values = np.logspace(0, 1, 10)
plot_bounds(N_values, d_vc = d_vc, delta = delta, N_target = N_target)
```




Now, we will find the smallest for very small $N$, say $N = 5$.

```
smallest_bound_name, smallest_bound_value = find_smallest_VC_bound(N_target, d_vc, delta)
print(f"The smallest VC Bound is {smallest_bound_name} with value {smallest_bound_value:.6f}.")
```

```
The smallest VC Bound is Parrondo and Van den Broek Bound with value 1.743954.
```

Finally, the correct answer is **[c]** Parrondo and Van den Broek: $\epsilon \leq \sqrt{\frac{1}{N}(2\epsilon + ln(\frac{6m_{\mathcal{H}}(2N)}{\delta}))}$

## Bias and Variance

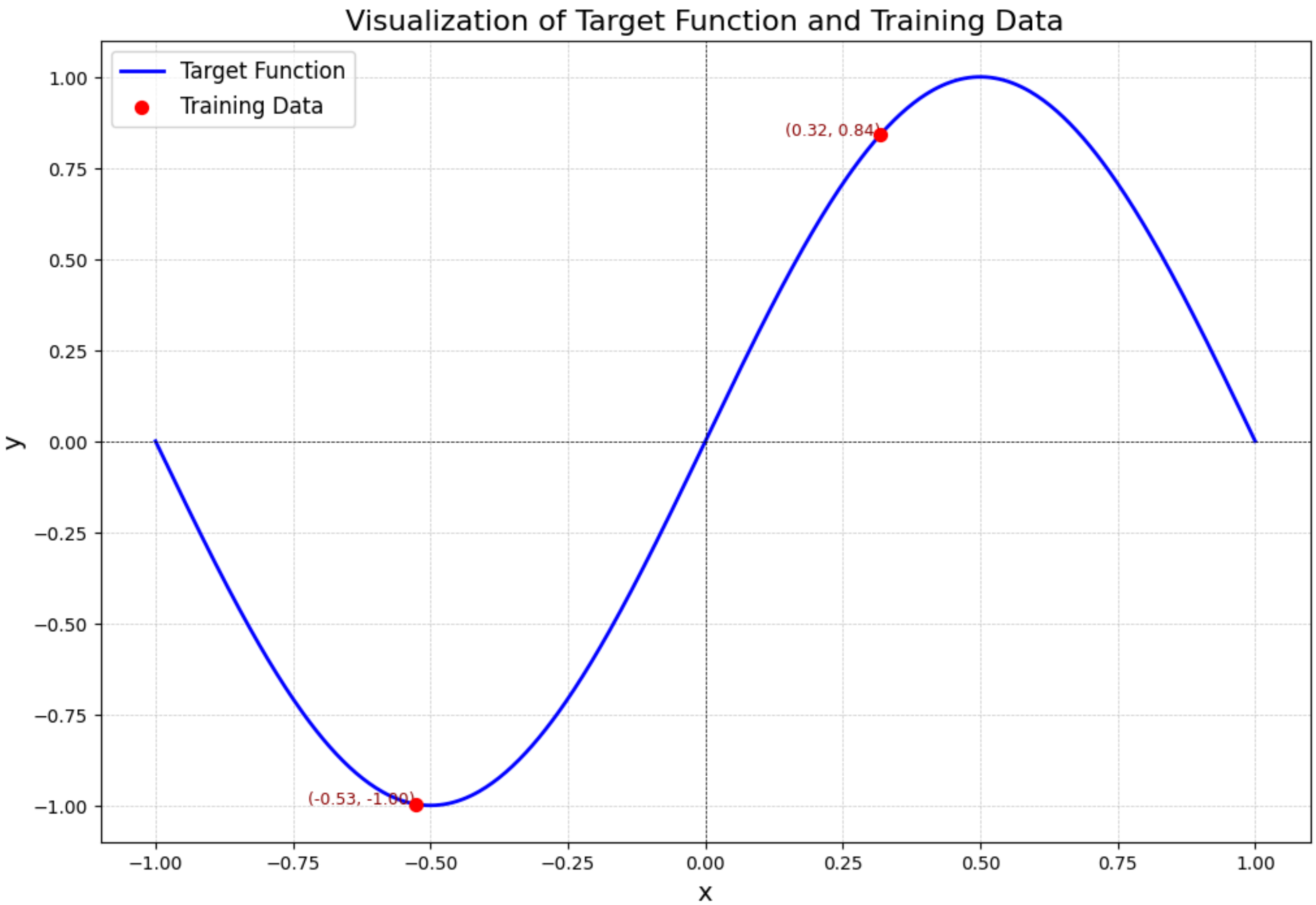
Consider the case where the target function $f : [-1, 1] \to \mathbb{R}$ is given by $f(x) = sin(\pi x)$ and the input probability distribution is uniform on $[-1, 1]$. Assume that the training set has only two examples (picked independently), and that the learning algorithm produces the hypothesis that minimizes the mean squared error on the examples.

- Building the class `TrainingDataset`

```
class TrainingDataset:
  def __init__(self, N: int = 2, lower_bound = -1.0, upper_bound = 1.0, target_function: Callable[[float], float] = None):
    self.N = N
    self.lower_bound = lower_bound
    self.upper_bound = upper_bound
    self.target_function = target_function if target_function else self.default_target_function

    # sinh ra bộ training dataset
    self.X = self.generate_inputs() # generating inputs
    self.y = self.evaluate_outputs() # evaluating outputs

  @staticmethod
  def default_target_function(x: float) -> float:
    return np.sin(np.pi * x)

  def generate_inputs(self) -> List[float]:
    return np.random.uniform(self.lower_bound, self.upper_bound, self.N).tolist()

  def evaluate_outputs(self) -> List[float]:
    return [self.target_function(x) for x in self.X]

  def get_training_data(self) -> List[tuple]:
    return list(zip(self.X, self.y))

  def display(self):
    print("Training Dataset:")
    for i, (x, y) in enumerate(self.get_training_data(), start = 1):
      print(f"Sample {i}: x = {x:.4f}, y = {y:.4f}")

  def visualize(self, resolution: int = 500):
    x_high_res = np.linspace(self.lower_bound, self.upper_bound, resolution)
    y_high_res = [self.target_function(x) for x in x_high_res]

    plt.figure(figsize = (12, 8))
    plt.plot(x_high_res, y_high_res, label="Target Function", color="blue", linewidth=2)

    plt.scatter(self.X, self.y, color="red", label="Training Data", s=50, zorder=5)

    for i, (x, y) in enumerate(self.get_training_data()):
      plt.text(x, y, f"({x:.2f}, {y:.2f})", fontsize=9, ha='right', color='darkred')

    plt.title("Visualization of Target Function and Training Data", fontsize=16)
    plt.xlabel("x", fontsize=14)
    plt.ylabel("y", fontsize=14)
    plt.axhline(0, color="black", linewidth=0.5, linestyle="--")
    plt.axvline(0, color="black", linewidth=0.5, linestyle="--")
    plt.grid(True, which="both", linestyle="--", linewidth=0.5, alpha=0.7)
    plt.legend(fontsize=12)
    plt.show()
```

```
training_dataset = TrainingDataset(N = 2)
training_dataset.display()
training_dataset.visualize()
```

```
Training Dataset:
Sample 1: x = 0.3181, y = 0.8411
Sample 2: x = -0.5275, y = -0.9963
```




## Problem $04$

Assume the learning model consists of all hypotheses of the form $h(x) = ax$. What is the expected value, $\overline{g}(x)$, of the hypothesis produced by the learning algorithm (expected value with respect to the data set)? Express your $\overline{g}(x)$ as $\hat{a}x$, and round $\hat{a}$ to two decimal digits only, then match *exactly* to one of the following answers.

**[a]** $\overline{g}(x) = 0$

**[b]** $\overline{g}(x) = 0.79x$

**[c]** $\overline{g}(x) = 1.07x$

**[d]** $\overline{g}(x) = 1.58x$

**[e]** None of the above

**Answer**

- Building the class `Hypothesis` to store all equation of each hypothesis.

```
class Hypothesis:
  TYPES: Dict[str, Callable[[float, List[float]], float]] = {
    "h(x) = b": lambda x, params: params[0],  #h(x) = b
    "h(x) = ax": lambda x, params: params[0] * x, #h(x) = ax
    "h(x) = ax + b": lambda x, params: params[0] * x + params[1], #h(x) = ax + b
    "h(x) = ax^2": lambda x, params: params[0] * x**2, #h(x) = ax^2
    "h(x) = ax^2 + b": lambda x, params: params[0] * x**2 + params[1] #h(x) = ax^2 + b
  }
```

- Building the class `HypothesisAnalysis` to implement linear regression for fitting a set of data points.

```python
class HypothesisAnalysis:
  def __init__(self, training_set: TrainingDataset, hypothesis_type: str):
    self.training_set = training_set
    self.hypothesis_type = hypothesis_type

  def prepare_feature(self, X: List[float]) -> np.ndarray:
    if self.hypothesis_type == "h(x) = b": # h(x) = b
      return np.ones((len(X), 1)) # chỉ có cột bias
    elif self.hypothesis_type == "h(x) = ax": # h(x) = ax
      return np.array([[x] for x in X]) # chỉ có cột x
    elif self.hypothesis_type == "h(x) = ax + b": # h(x) = ax + b
      return np.array([[x, 1] for x in X]) # thêm cột bias vào từng phần tử X
    elif self.hypothesis_type == "h(x) = ax^2": # h(x) = ax^2
      return np.array([[x**2] for x in X]) # chỉ có cột x^2
    elif self.hypothesis_type == "h(x) = ax^2 + b": # h(x) = ax^2 + b
      return np.array([[x**2, 1] for x in X]) # thêm cột bias vào từng phần tử X
    else:
      raise ValueError(f"Unsupported hypothesis type: {self.hypothesis_type}")

  def fit_hypothesis(self) -> List[float]:
    X = self.prepare_feature(self.training_set.X)
    y = np.array(self.training_set.y)

    X_pinv = np.linalg.pinv(X)
    params = np.dot(X_pinv, y)
    return params.tolist()

  def predict_hypothesis(self, x: float, params: List[float]) -> float:
    hypothesis_value = Hypothesis.TYPES[self.hypothesis_type](x, params)
    return hypothesis_value
```

- Building the class `BiasVarianceAnalysis`:
  - Function `compute_gbar_params`: we will find all parameters of the `average` hypothesis $\bar{g}(\mathbf{x})$
$$\bar{g}(\mathbf{x}) = \mathbb{E}_{\mathcal{D}}[\mathbf{g}^{(\mathcal{D})}(\mathbf{x})]$$
  - In **many** data sets $\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_k$, the `average` hypothesis is
$$\bar{g}(\mathbf{x}) \approx \frac{\mathbf{1}}{\mathbf{K}} \sum_{\mathbf{k}=\mathbf{1}}^{\mathbf{K}} \mathbf{g}^{(\mathcal{D}_\mathrm{k})}(\mathbf{x})$$

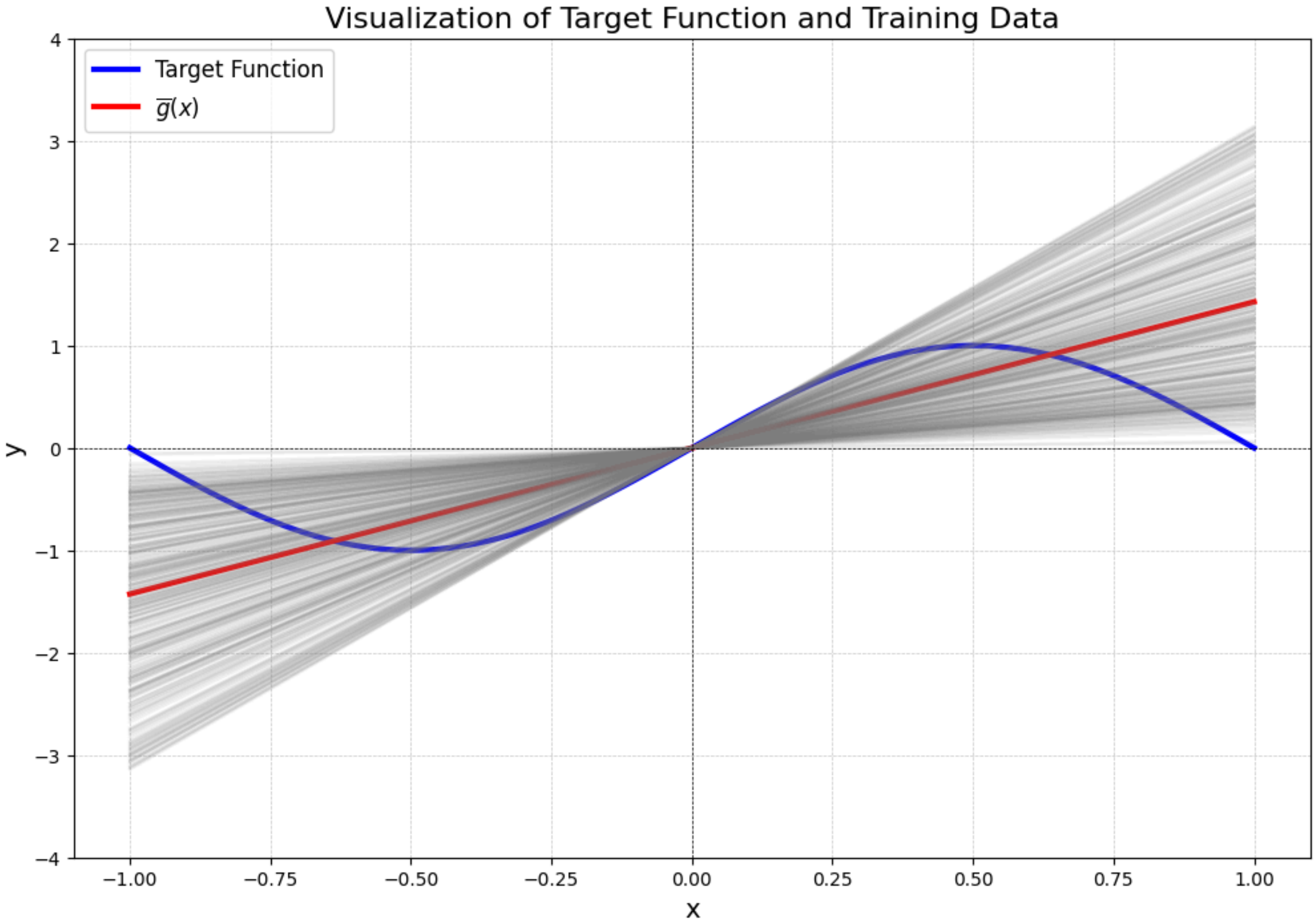
```python
class BiasVarianceAnalysis:
  def __init__(self, hypothesis_type: str, num_simulations: int = 100000, N: int = 2, target_function: Callable[[float], float] = None):
    self.hypothesis_type = hypothesis_type
    self.N = N
    self.num_simulations = num_simulations
    self.target_function = target_function if target_function else TrainingDataset.default_target_function
    self.gbar_params = None
    self.g_params_sets = []
    self.training_sets = []
    self.bias = 0
    self.variance = 0
    self.eout = 0

  def compute_gbar_params(self):
    self.g_params_sets = []
    self.training_sets = []

    for _ in range(self.num_simulations):
      # Sinh tập dữ liệu huấn luyện
      training_dataset = TrainingDataset(N=self.N, target_function=self.target_function)
      hypothesis_analysis = HypothesisAnalysis(training_dataset, self.hypothesis_type)
      params = hypothesis_analysis.fit_hypothesis()
      self.g_params_sets.append(params)
      self.training_sets.append(training_dataset)

    self.gbar_params = np.mean(np.array(self.g_params_sets), axis = 0).tolist()

    if self.hypothesis_type == "h(x) = b":
      print(f"The expected value gbar(x) of hypothesis {self.hypothesis_type} produced by learning algorithm is:")
      print(f"g_bar(x) = {self.gbar_params[0]:.2f}")
    elif self.hypothesis_type == "h(x) = ax":
      print(f"The expected value gbar(x) of hypothesis {self.hypothesis_type} produced by learning algorithm is:")
      print(f"gbar(x) = {self.gbar_params[0]:.2f}x")
    elif self.hypothesis_type == "h(x) = ax + b":
```

```
  elif self.hypothesis_type == "h(x) = ax + b":
   print(f"The expected value gbar(x) of hypothesis {self.hypothesis_type} produced by learning algorithm is:")
   print(f"gbar(x) = {self.gbar_params[0]:.2f}x + {self.gbar_params[1]:.2f}")
  elif self.hypothesis_type == "h(x) = ax^2":
   print(f"The expected value gbar(x) of hypothesis {self.hypothesis_type} produced by learning algorithm is:")
   print(f"gbar(x) = {self.gbar_params[0]:.2f}x^2")
  elif self.hypothesis_type == "h(x) = ax^2 + b":
   print(f"The expected value gbar(x) of hypothesis {self.hypothesis_type} produced by learning algorithm is:")
   print(f"gbar(x) = {self.gbar_params[0]:.2f}x^2 + {self.gbar_params[1]:.2f}")
  else:
   raise ValueError(f"Unsupported hypothesis type: {self.hypothesis_type}")

 def compute_bias_and_variance(self):
   # Tạo tập kiểm tra độc lập
   test_x = np.linspace(self.training_sets[0].lower_bound, self.training_sets[0].upper_bound, 100)
   test_y = np.array([self.target_function(x) for x in test_x])

   # Tính g_bar(x) trên tập kiểm tra
   gbar_test = np.array([
     Hypothesis.TYPES[self.hypothesis_type](x, self.gbar_params) for x in test_x
   ])

   # Tính bias: khoảng cách trung bình giữa gbar(x) và f(x)
   self.bias = np.mean((gbar_test - test_y)**2)

   # Tính variance: khoảng cách giữa g(x) và gbar(x) trên tập kiểm tra
   variances = []
   for params in self.g_params_sets:
     g_test = np.array([
       Hypothesis.TYPES[self.hypothesis_type](x, params) for x in test_x
     ])
     variances.append((g_test - gbar_test)**2)
   self.variance = np.mean(variances)

   # Tổng lỗi đầu ra
   self.eout = self.bias + self.variance

 def visualize(self, resolution: int = 500, y_limit: tuple = (-4, 4)):
  x_vals = np.linspace(-1, 1, resolution)
  y_vals = [self.target_function(x) for x in x_vals]
  gbar_vals = [Hypothesis.TYPES[self.hypothesis_type](x, self.gbar_params) for x in x_vals]

  plt.figure(figsize=(12, 8))
  plt.plot(x_vals, y_vals, label="Target Function", color="blue", linewidth=3)
  plt.plot(x_vals, gbar_vals, label="$\overline{g}(x)$", color="red", linewidth=3)

  # Plot individual hypotheses with clipping
  for params in self.g_params_sets[:400]:  # Limit to 400 hypotheses for clarity
    hypothesis_vals = [Hypothesis.TYPES[self.hypothesis_type](x, params) for x in x_vals]
    hypothesis_vals_clipped = np.clip(hypothesis_vals, y_limit[0], y_limit[1])  # Clip y-values
    plt.plot(x_vals, hypothesis_vals_clipped, color="gray", alpha=0.1)

  plt.title("Visualization of Target Function and Training Data", fontsize=16)
  plt.xlabel("x", fontsize=14)
  plt.ylabel("y", fontsize=14)
  plt.axhline(0, color="black", linewidth=0.5, linestyle="--")
  plt.axvline(0, color="black", linewidth=0.5, linestyle="--")
  plt.ylim(y_limit)  # Set y-axis limits
  plt.grid(True, which="both", linestyle="--", linewidth=0.5, alpha=0.7)
  plt.legend(fontsize=12)
  plt.show()

 def result(self):
  print(f"Analyzing Hypothesis {self.hypothesis_type} ... \n")
  self.compute_gbar_params()
  self.compute_bias_and_variance()
  print(f"The bias of hypothesis {self.hypothesis_type} produced by learning algorithm is: {self.bias:.2f}")
  print(f"The variance of hypothesis {self.hypothesis_type} produced by learning algorithm is: {self.variance:.2f}")
  print(f"The out-of-sample error of hypothesis {self.hypothesis_type} produced by learning algorithm is: {self.eout:.2f}")
  self.visualize()
```

```
hypothesis_type = "h(x) = ax"

bias_variance_analysis = BiasVarianceAnalysis(
  hypothesis_type = hypothesis_type,
  N = 2,
  num_simulations = 100000
```

```
)

bias_variance_analysis.result()
```

```
Analyzing Hypothesis h(x) = ax ...

The expected value gbar(x) of hypothesis h(x) = ax produced by learning algorithm is:
gbar(x) = 1.43x
The bias of hypothesis h(x) = ax produced by learning algorithm is: 0.29
The variance of hypothesis h(x) = ax produced by learning algorithm is: 0.24
The out-of-sample error of hypothesis h(x) = ax produced by learning algorithm is: 0.53
```




So, we can see the expected value $\overline{g}(x)$ of hypothesis produced by the learning algorithm is $\overline{g}(x) = 1.42x$.

As the answer does not *exactly* match any of the given above solutions. So we will choose answer **[e]**.

Finally, the correct answer is **[e]** None of the above.

### Problem $05$

What is the closest value to the bias in this case?

**[a]** $0.1$

**[b]** $0.3$

**[c]** $0.5$

**[d]** $0.7$

**[e]** $1.0$

**Answer**

- Function `compute_bias_and_variance`:

- $\text{bias} = \mathbb{E}_{\mathbf{x}}[(\bar{g}(\mathbf{x}) - \mathbf{f}(\mathbf{x})^2]$
- $\text{var} = \mathbb{E}_{\mathbf{x}}[\mathbb{E}_{\mathcal{D}}[g^{(\mathcal{D})}(\mathbf{x}) - \bar{\mathbf{g}}(\mathbf{x})^2]]$
- $\mathbb{E}_{\mathcal{D}}[\mathbb{E}_{\text{out}}(g^{(\mathcal{D})})] = \text{bias} + \text{var}$

```
choices = [0.1, 0.3, 0.5, 0.7, 1.0]
bias = bias_variance_analysis.bias
nearest_answer, result = find_nearest_answer(bias, choices)
print(f"The nearest answer among the given choices is {nearest_answer} with the final result {result}.")
```

```
The nearest answer among the given choices is 0.3 with the final result 0.2884138622926367.
```

Finally, the correct answer is **[b]** $0.3$

## Problem $06$

What is the closest value to the variance in this case?

**[a]** $0.2$

**[b]** $0.4$

**[c]** $0.6$

**[d]** $0.8$

**[e]** $1.0$

**Answer**

```
choices = [0.2, 0.4, 0.6, 0.8, 1.0]
var = bias_variance_analysis.variance
nearest_answer, result = find_nearest_answer(var, choices)
print(f"The nearest answer among the given choices is {nearest_answer} with the final result {result}.")
```

```
The nearest answer among the given choices is 0.2 with the final result 0.24151752619744968.
```

Finally, the correct answer is **[a]** $0.2$

## Problem $07$

Now, let's change $\mathcal{H}$. Which of the following learning models has the least expected value of out-of-sample error?

**[a]** Hypotheses of the form $h(x) = b$

**[b]** Hypotheses of the form $h(x) = ax$

**[c]** Hypotheses of the form $h(x) = ax + b$

**[d]** Hypotheses of the form $h(x) = ax^2$

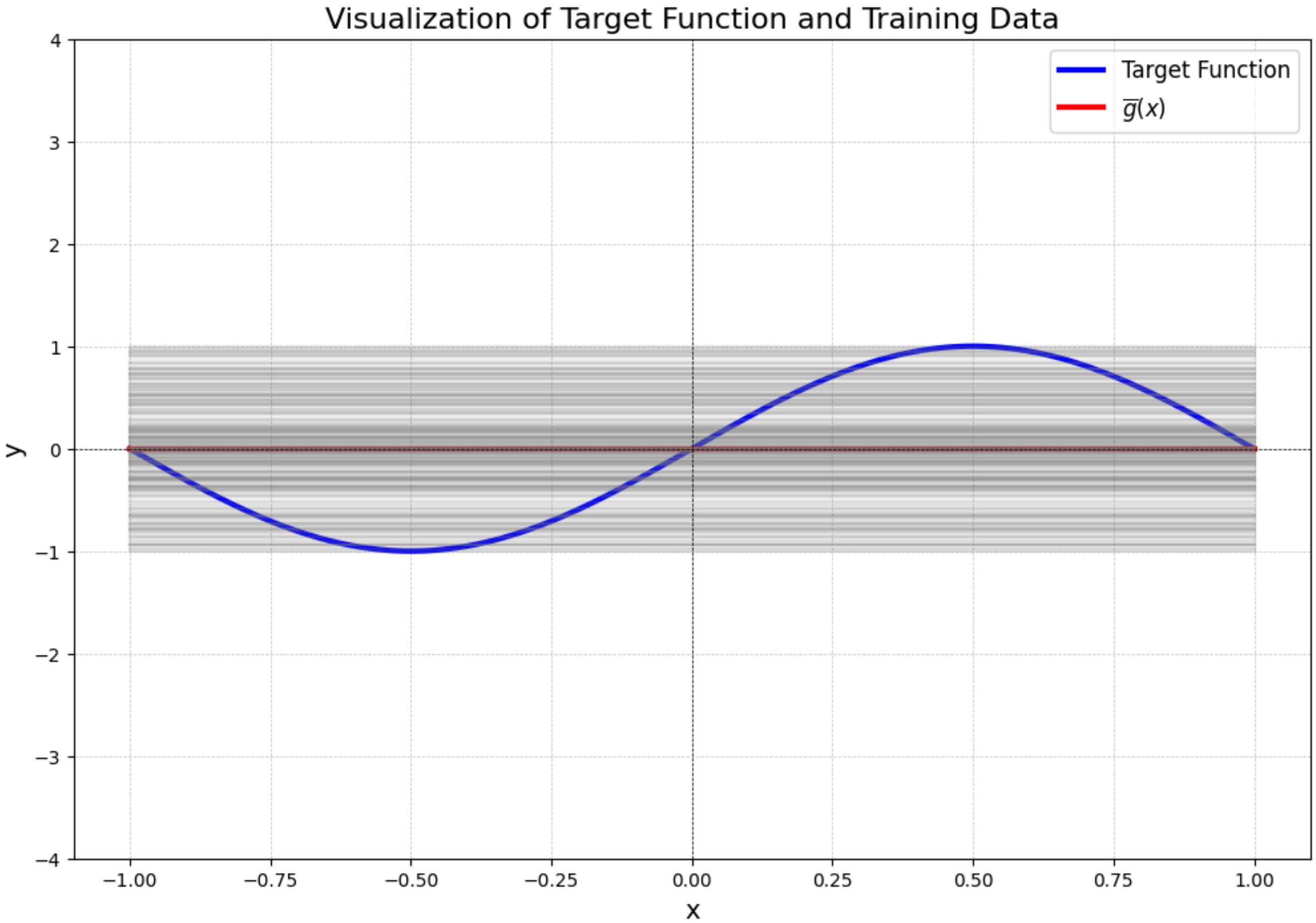
**[e]** Hypotheses of the form $h(x) = ax^2 + b$

**Answer**

**[a]** Hypotheses of the form $h(x) = b$

```
hypothesis_type = "h(x) = b"

bias_variance_analysis_1 = BiasVarianceAnalysis(
    hypothesis_type = hypothesis_type,
    N = 2,
    num_simulations = 100000
)

bias_variance_analysis_1.result()
```

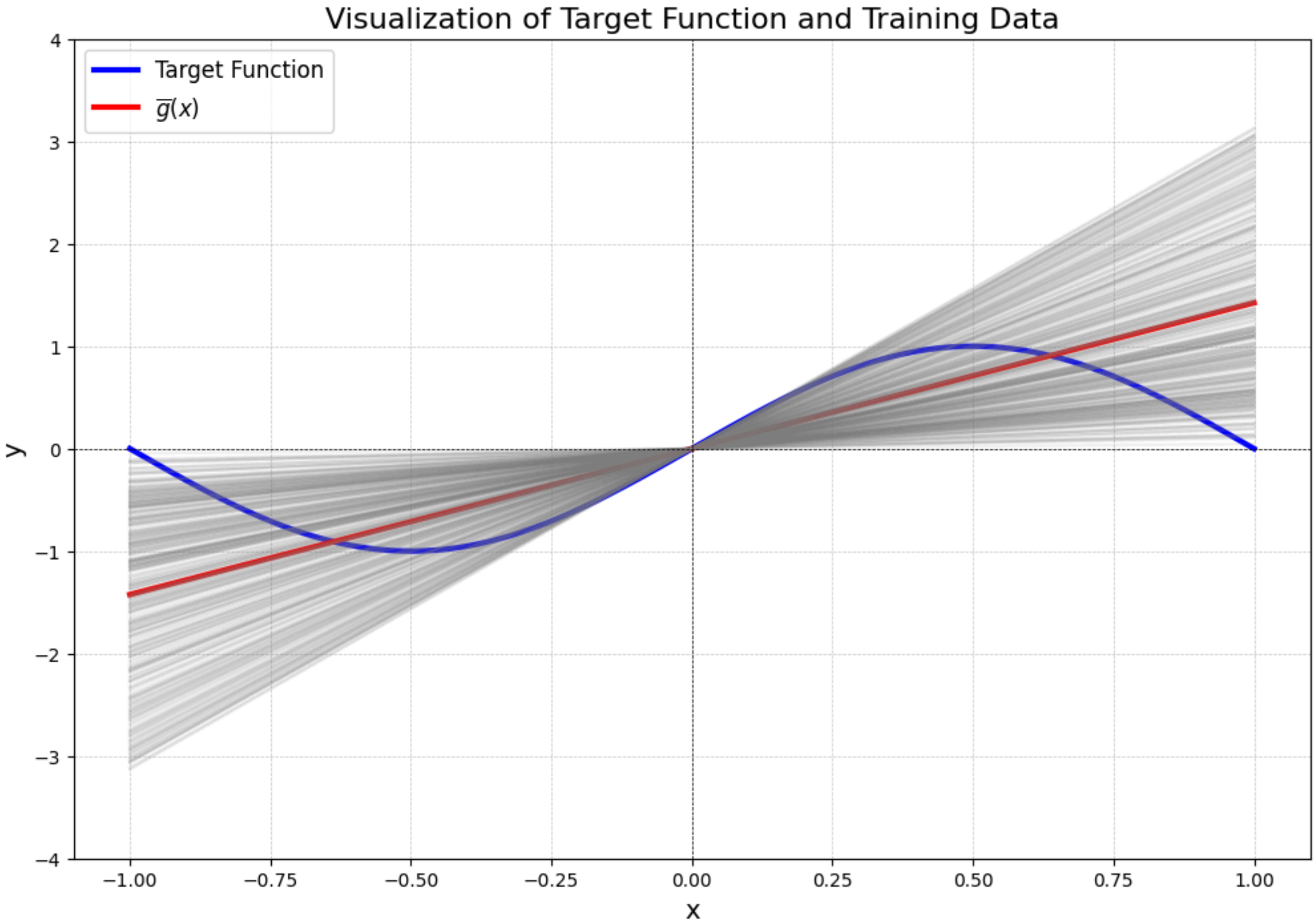
```
Analyzing Hypothesis h(x) = b ...

The expected value gbar(x) of hypothesis h(x) = b produced by learning algorithm is:
g_bar(x) = 0.00
The bias of hypothesis h(x) = b produced by learning algorithm is: 0.50
The variance of hypothesis h(x) = b produced by learning algorithm is: 0.25
The out-of-sample error of hypothesis h(x) = b produced by learning algorithm is: 0.74
```




**[b]** Hypotheses of the form $h(x) = ax$

```
hypothesis_type = "h(x) = ax"

bias_variance_analysis_2 = BiasVarianceAnalysis(
    hypothesis_type = hypothesis_type,
    N = 2,
    num_simulations = 100000
)

bias_variance_analysis_2.result()
```

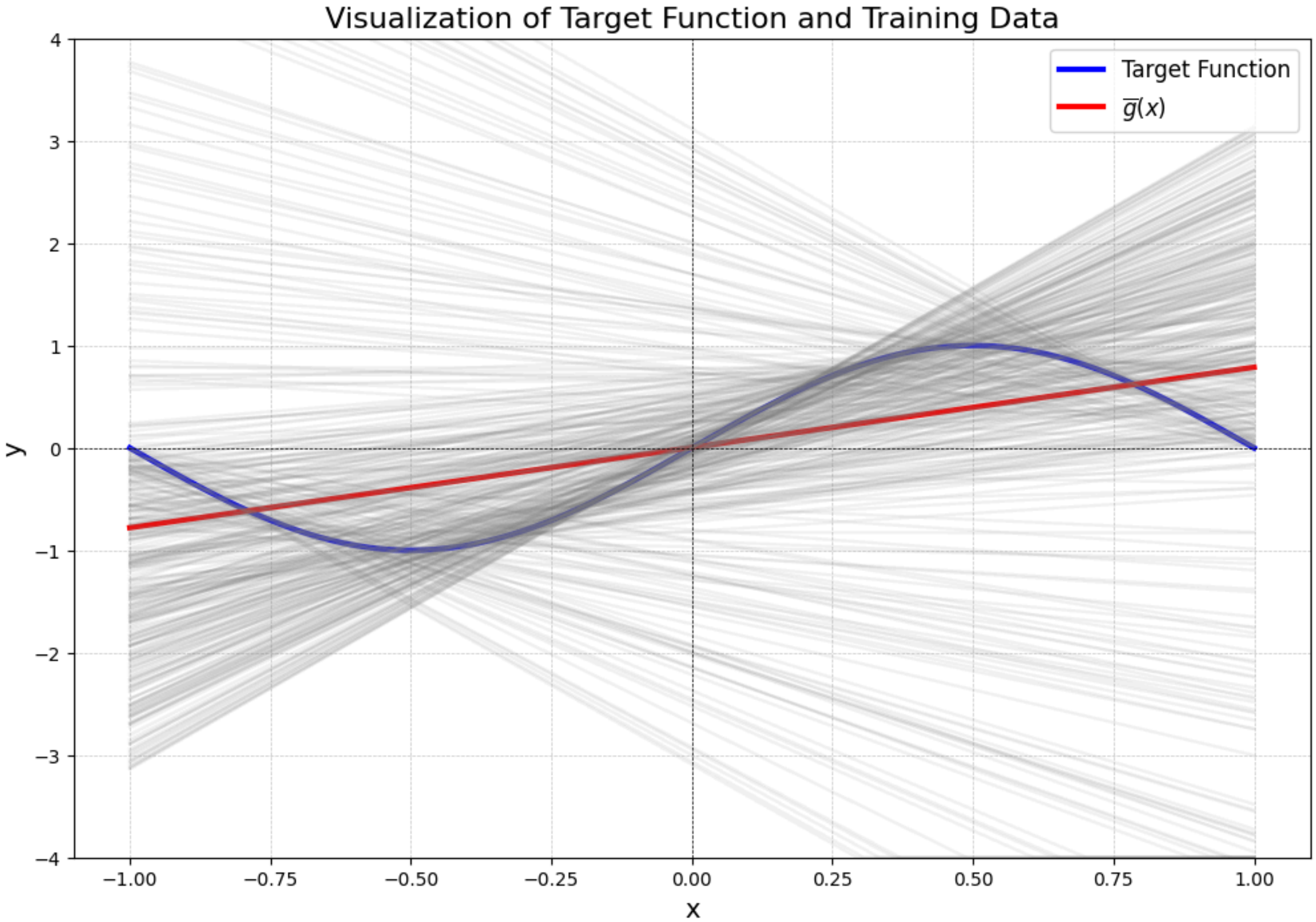
```
Analyzing Hypothesis h(x) = ax ...

The expected value gbar(x) of hypothesis h(x) = ax produced by learning algorithm is:
gbar(x) = 1.42x
The bias of hypothesis h(x) = ax produced by learning algorithm is: 0.29
The variance of hypothesis h(x) = ax produced by learning algorithm is: 0.24
The out-of-sample error of hypothesis h(x) = ax produced by learning algorithm is: 0.53
```




**[c]** Hypotheses of the form $h(x) = ax + b$

```
hypothesis_type = "h(x) = ax + b"

bias_variance_analysis_3 = BiasVarianceAnalysis(
    hypothesis_type = hypothesis_type,
    N = 2,
    num_simulations = 100000
)

bias_variance_analysis_3.result()
```

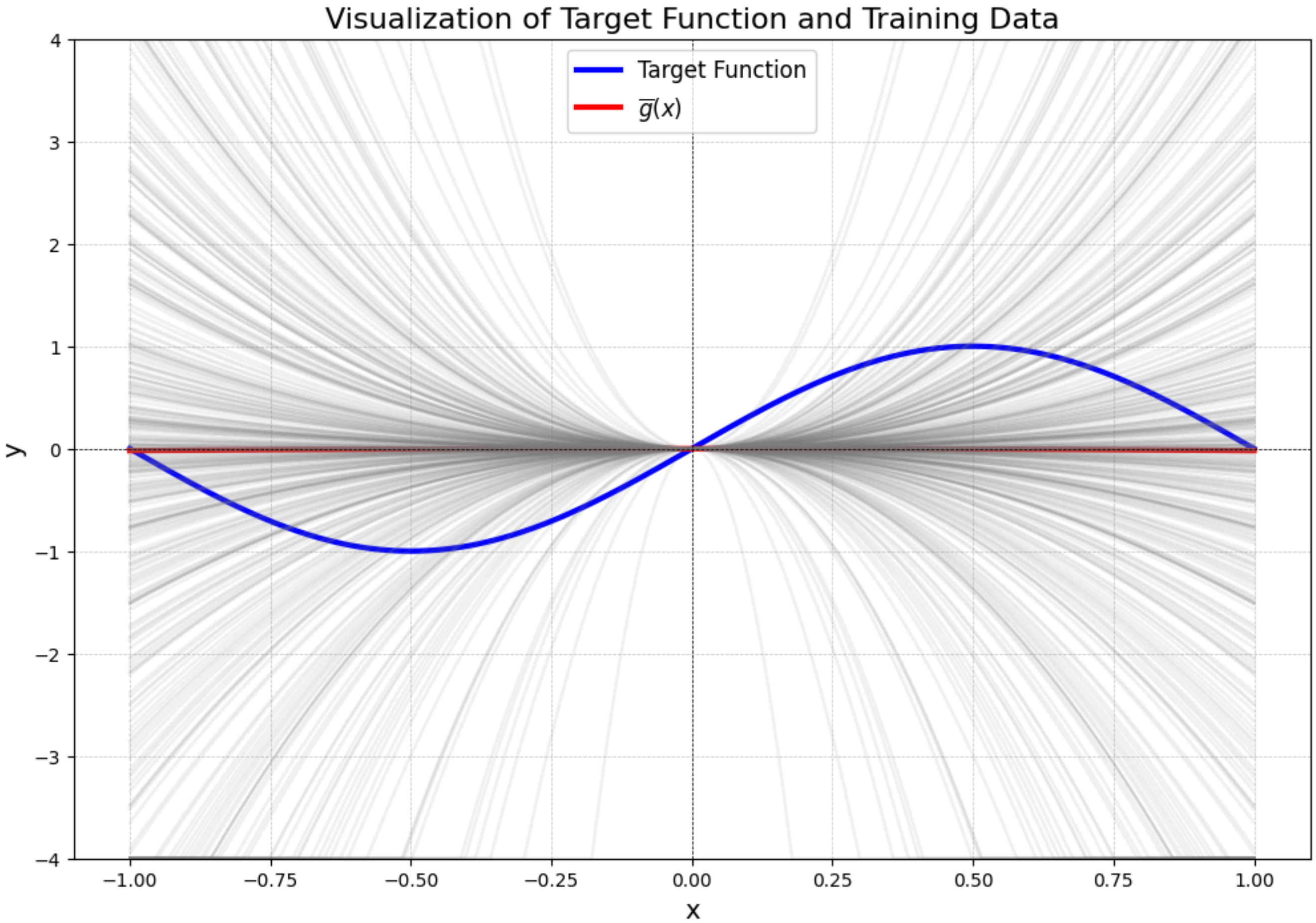
```
Analyzing Hypothesis h(x) = ax + b ...

The expected value gbar(x) of hypothesis h(x) = ax + b produced by learning algorithm is:
gbar(x) = 0.78x + 0.00
The bias of hypothesis h(x) = ax + b produced by learning algorithm is: 0.21
The variance of hypothesis h(x) = ax + b produced by learning algorithm is: 1.68
The out-of-sample error of hypothesis h(x) = ax + b produced by learning algorithm is: 1.89
```




**[d]** Hypotheses of the form $h(x) = ax^2$

```
hypothesis_type = "h(x) = ax^2"

bias_variance_analysis_4 = BiasVarianceAnalysis(
    hypothesis_type = hypothesis_type,
    N = 2,
    num_simulations = 100000
)

bias_variance_analysis_4.result()
```

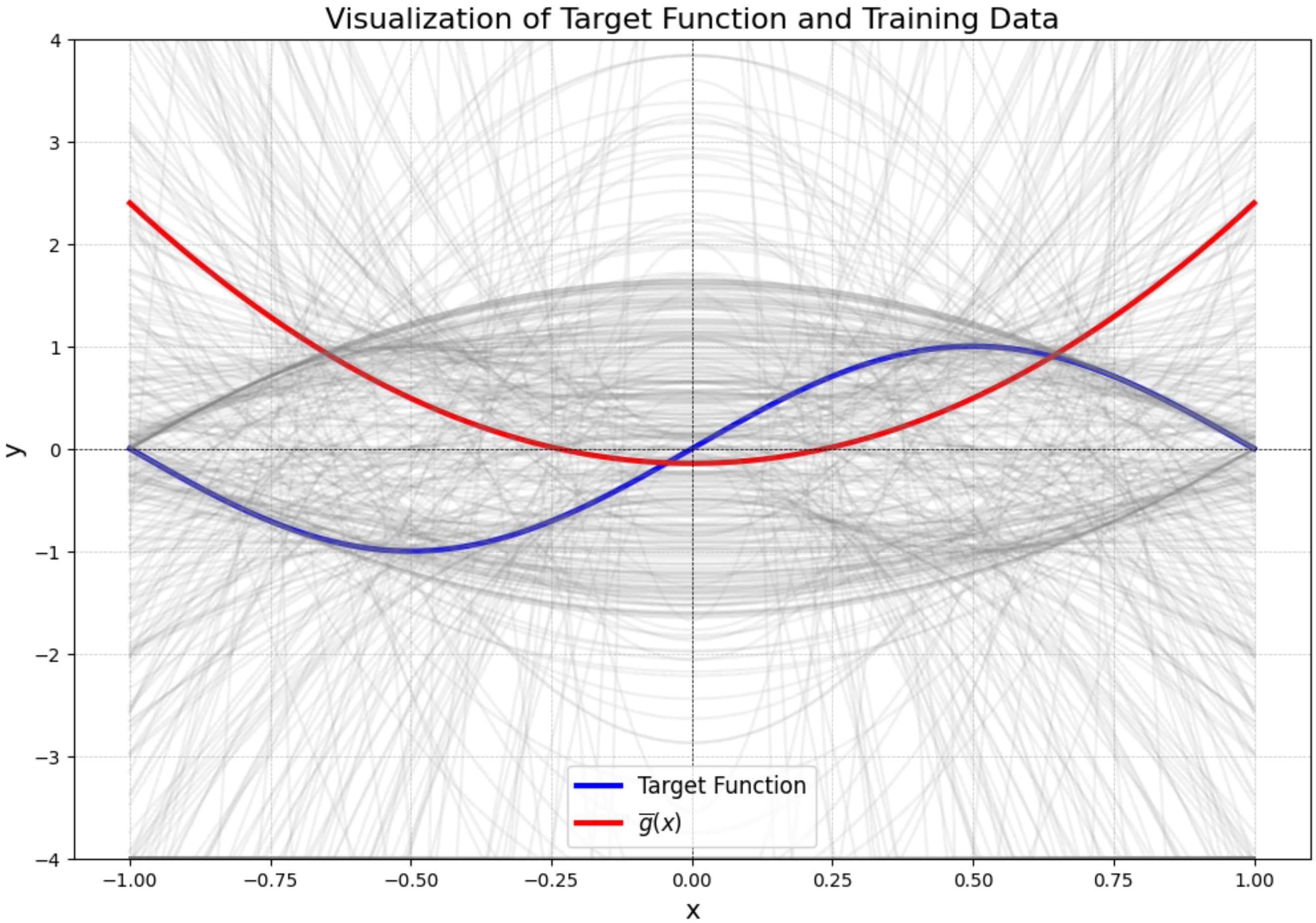
```
Analyzing Hypothesis h(x) = ax^2 ...

The expected value gbar(x) of hypothesis h(x) = ax^2 produced by learning algorithm is:
gbar(x) = -0.02x^2
The bias of hypothesis h(x) = ax^2 produced by learning algorithm is: 0.50
The variance of hypothesis h(x) = ax^2 produced by learning algorithm is: 16.84
The out-of-sample error of hypothesis h(x) = ax^2 produced by learning algorithm is: 17.34
```




**[e]** Hypotheses of the form $h(x) = ax^2 + b$

```
hypothesis_type = "h(x) = ax^2 + b"

bias_variance_analysis_5 = BiasVarianceAnalysis(
    hypothesis_type = hypothesis_type,
    N = 2,
    num_simulations = 100000
)

bias_variance_analysis_5.result()
```

```
Analyzing Hypothesis h(x) = ax^2 + b ...

The expected value gbar(x) of hypothesis h(x) = ax^2 + b produced by learning algorithm is:
gbar(x) = 2.54x^2 + -0.14
The bias of hypothesis h(x) = ax^2 + b produced by learning algorithm is: 1.61
The variance of hypothesis h(x) = ax^2 + b produced by learning algorithm is: 75779.09
The out-of-sample error of hypothesis h(x) = ax^2 + b produced by learning algorithm is: 75780.70
```




```
print(f'Expected value of out-of-sample error for h(x) = b is {bias_variance_analysis_1.eout:.2f}')
print(f'Expected value of out-of-sample error for h(x) = ax is {bias_variance_analysis_2.eout:.2f}')
print(f'Expected value of out-of-sample error for h(x) = ax+b is {bias_variance_analysis_3.eout:.2f}')
print(f'Expected value of out-of-sample error for h(x) = ax^2 is {bias_variance_analysis_4.eout:.2f}')
print(f'Expected value of out-of-sample error for h(x) = ax^2+b is {bias_variance_analysis_5.eout:.2f}')
```

```
Expected value of out-of-sample error for h(x) = b is 0.74
Expected value of out-of-sample error for h(x) = ax is 0.53
Expected value of out-of-sample error for h(x) = ax+b is 1.89
Expected value of out-of-sample error for h(x) = ax^2 is 17.34
Expected value of out-of-sample error for h(x) = ax^2+b is 75780.70
```

So, the least expected value of out-of-sample error has the hypothesis $h(x) = ax$ with the error at $0.53$.

Finally, the correct answer is **[b]** Hypotheses of the form $h(x) = ax$

## VC Dimension

### Problem $08$

Assume $q \geq 1$ is an integer and let $m_{\mathcal{H}}(1) = 2$. What is the VC dimension of a hypothesis set whose growth function satisfies: $m_{\mathcal{H}}(N+1) = 2m_{\mathcal{H}}(N) - \binom{N}{q}$ ? Recall that $\binom{M}{m} = 0$ when $m > M$.

**[a]** $q - 2$

**[b]** $q - 1$

**[c]** $q$

**[d]** $q + 1$

**[e]** None of the above

**Answer**

$q \geq 1 \in \mathbb{Z}$ , $m_{\mathcal{H}}(1) = 2$ $\binom{M}{m} = 0$, $m > M$

Finally, the correct answer is **[c]** $q$.

## Problem 09

For hypothesis sets $\mathcal{H}_1, \mathcal{H}_2, \ldots, \mathcal{H}_K$ with finite, positive VC dimensions $d_{\mathrm{VC}}(\mathcal{H}_k)$, some of the following bounds are correct and some are not. Which among the correct ones is the tightest bound (the smallest range of values) on the VC dimension of the **intersection** of the sets: $d_{\mathrm{VC}}\left(\bigcap_{k=1}^{K} \mathcal{H}_k\right)$ ? (The VC dimension of an empty set or a singleton set is taken as zero)

**[a]** $0 \leq d_{\mathrm{VC}}\left(\bigcap_{k=1}^{K} \mathcal{H}_k\right) \leq \sum_{k=1}^{K} d_{\mathrm{VC}}(\mathcal{H}_k)$

**[b]** $0 \leq d_{\mathrm{VC}}\left(\bigcap_{k=1}^{K} \mathcal{H}_k\right) \leq \min\{d_{\mathrm{VC}}(\mathcal{H}_k)\}_{k=1}^{K}$

**[c]** $0 \leq d_{\mathrm{VC}}\left(\bigcap_{k=1}^{K} \mathcal{H}_k\right) \leq \max\{d_{\mathrm{VC}}(\mathcal{H}_k)\}_{k=1}^{K}$

**[d]** $\min\{d_{\mathrm{VC}}(\mathcal{H}_k)\}_{k=1}^{K} \leq d_{\mathrm{VC}}\left(\bigcap_{k=1}^{K} \mathcal{H}_k\right) \leq \max\{d_{\mathrm{VC}}(\mathcal{H}_k)\}_{k=1}^{K}$

**[e]** $\min\{d_{\mathrm{VC}}(\mathcal{H}_k)\}_{k=1}^{K} \leq d_{\mathrm{VC}}\left(\bigcap_{k=1}^{K} \mathcal{H}_k\right) \leq \sum_{k=1}^{K} d_{\mathrm{VC}}(\mathcal{H}_k)$

(9)

1. We assume all of hypothesis $\mathcal{H}_1, \mathcal{H}_2, \ldots, \mathcal{H}_K$ sets are disjaint

$\Rightarrow d_{vc}\left(\bigcap_{k=1}^{K} \mathcal{H}_k\right) = 0$

Thus, the left baund of the VC dimension is $0 \leq d_{vc}\left(\bigcap_{k=1}^{K} \mathcal{H}_k\right)$ (1)

2. Let $p = \min\{d_{vc}(\mathcal{H}_k)\}_{k=1}^{K}$

$\mathcal{H}_i$ is the Hypothesis that has $d_{vc} = p \Rightarrow d_{vc}(\mathcal{H}_i) = p$ (2)

The fact: We will assume that the max value in $d_{vc}\left(\bigcap_{k=1}^{K} \mathcal{H}_k\right) > p$.

This means that the breakpoint is $k = p+1 \Rightarrow$ all of K hypothesis sets can be able to shatter more than p points.

$\Rightarrow \mathcal{H}_i$ can also be able to shatter more than p points $\Rightarrow d_{vc}(\mathcal{H}_i) > p$ (3)

(2), (3) $\Rightarrow$ contradict the fact that we assumed $\Rightarrow$ wrong.