Problem 1: Bias-Variance Decomposition

Part 1:

Let $y(x) = f(x) + \epsilon$, where $\epsilon \sim \mathcal{N}(0, \sigma^2)$ is the noise. The model's prediction is $\hat{y}(x) = g(x)$. The mean squared error (MSE) over test instances x_i is:

$$MSE = \frac{1}{t} \sum_{i=1}^{t} (f(x_i) + \epsilon_i - g(x_i))^2$$

Expanding:

$$MSE = \mathbb{E}\left[(f(x) + \epsilon - g(x))^2\right]$$

This decomposes as:

$$\mathbb{E}[\text{MSE}] = \underbrace{(\mathbb{E}[g(x)] - f(x))^2}_{\text{Bias}^2} + \underbrace{\mathbb{E}[(g(x) - \mathbb{E}[g(x)])^2]}_{\text{Variance}} + \underbrace{\mathbb{E}[\epsilon^2]}_{\text{Noise}}$$

Thus:

$$\mathbb{E}[\text{MSE}] = \text{Bias}^2 + \text{Variance} + \sigma^2$$

Part 2:

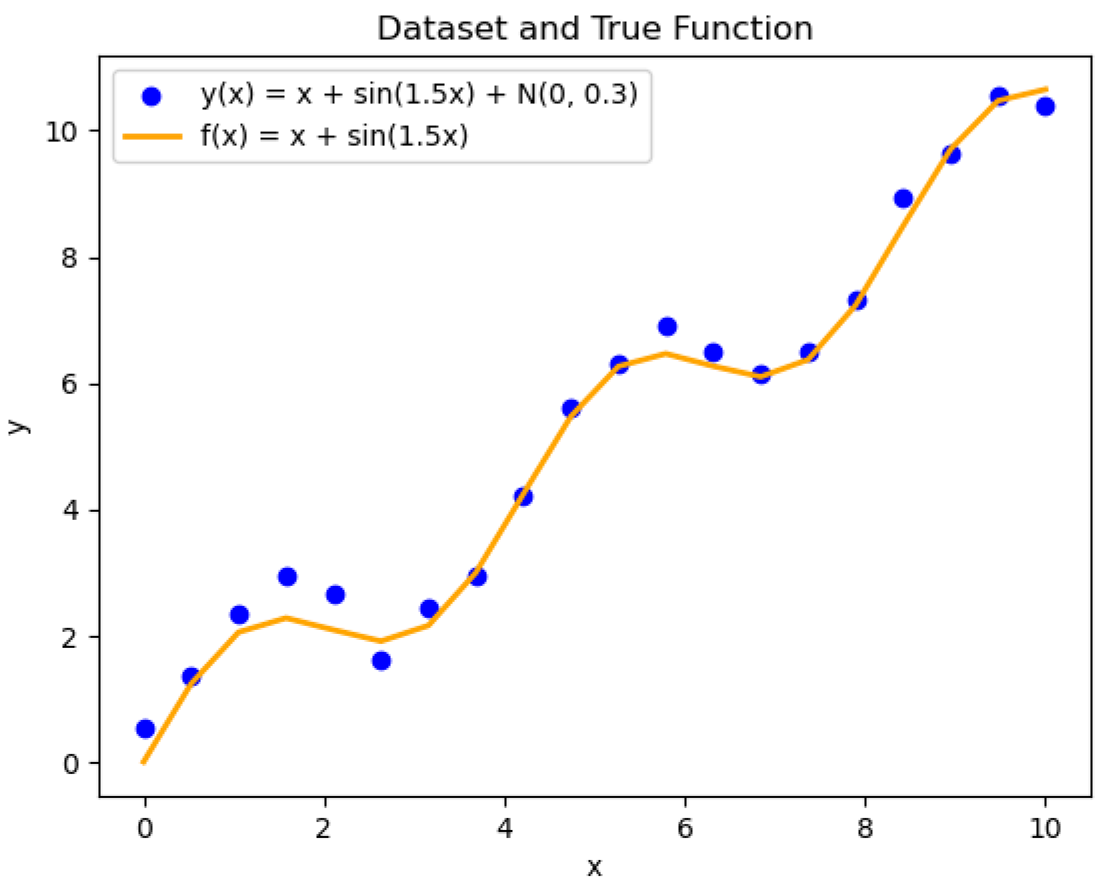
Given

$$y(x) = x + \sin(1.5x) + \mathcal{N}(0, 0.3)$$

Generate 20 points from y and display the dataset and f(x)

In [ ]:
```python
import numpy as np
import matplotlib.pyplot as plt

# Set random seed for reproducibility
np.random.seed(0)

# Generate x values in the range [0, 10]
x = np.linspace(0, 10, 20)

# Generate y values with noise
y = x + np.sin(1.5 * x) + np.random.normal(0, 0.3, size=x.shape)

# Define the true function f(x)
f_x = x + np.sin(1.5 * x)
```

```
# Plot y(x) as a scatter plot and f(x) as a smooth line
plt.scatter(x, y, label="y(x) = x + sin(1.5x) + N(0, 0.3)", color='blue')
plt.plot(x, f_x, label="f(x) = x + sin(1.5x)", color='orange', linewidth=2)
plt.xlabel("x")
plt.ylabel("y")
plt.legend()
plt.title("Dataset and True Function")
plt.show()
```



Part 3:

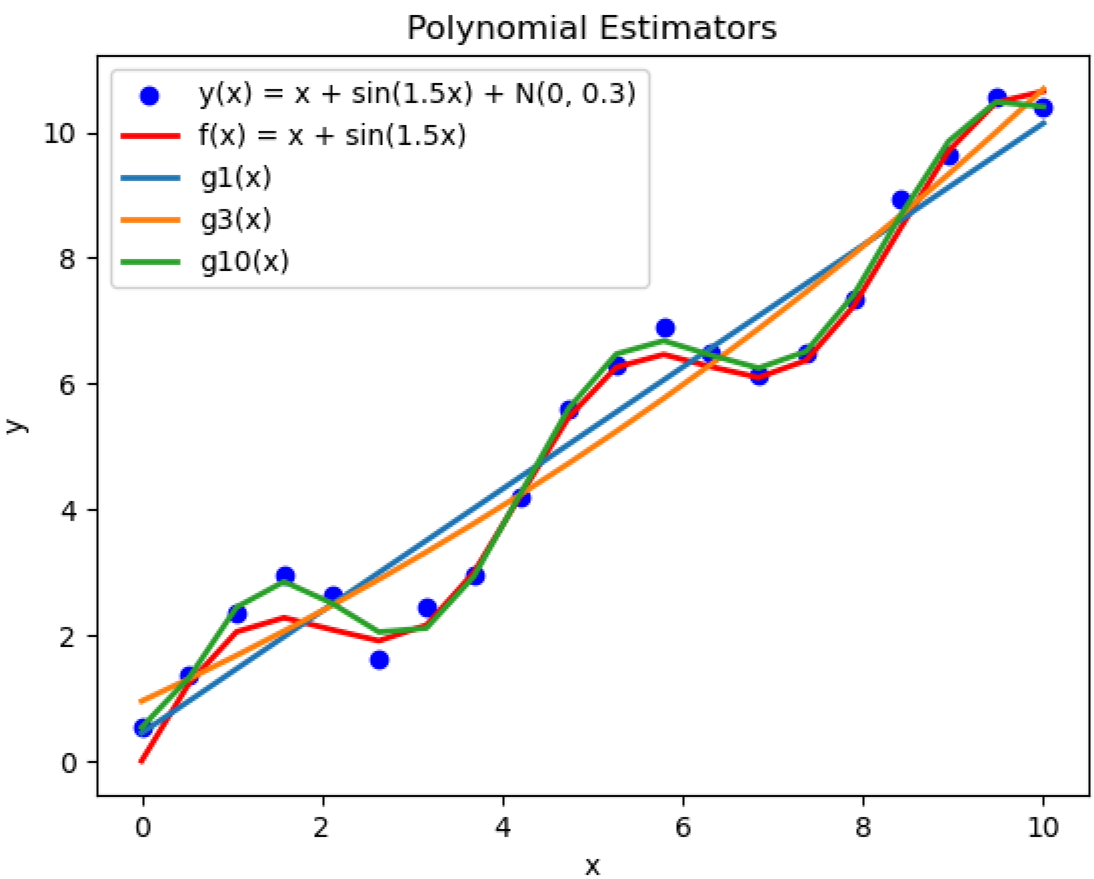
Use a weighted sum of polynomials as an estimator function for (f(x)). In particular, let the form of the estimator function be:

$$g_n(x) = \beta_0 + \beta_1 x + \beta_2 x^2 + \ldots + \beta_n x^n$$

Consider three candidate estimators, (g_1), (g_3), and (g_{10}). Estimate the coefficients of each of the three estimators using the sampled dataset and plot (y(x)), (f(x)), (g_1(x)), (g_3(x)), and (g_{10}(x)). Which estimator is underfitting? Which one is overfitting?

In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt

from sklearn.preprocessing import PolynomialFeatures
```

```
from sklearn.linear_model import LinearRegression
from sklearn.pipeline import make_pipeline

degrees = [1, 3, 10]
np.random.seed(0)
x = np.linspace(0, 10, 20)
x = x[:, np.newaxis]
y = x + np.sin(1.5 * x) + np.random.normal(0, 0.3, size=x.shape)
f_x = x + np.sin(1.5 * x)

plt.scatter(x, y, label="y(x) = x + sin(1.5x) + N(0, 0.3)", color='blue')
plt.plot(x, f_x, label="f(x) = x + sin(1.5x)", color='red', linewidth=2)


for degree in degrees:
    model = make_pipeline(PolynomialFeatures(degree), LinearRegression())
    model.fit(x, y)
    y_pred = model.predict(x[:])
    plt.plot(x, y_pred, label=f'g{degree}(x)', linewidth=2)

plt.xlabel("x")
plt.ylabel("y")
plt.legend()
plt.title("Polynomial Estimators")
plt.show()
```

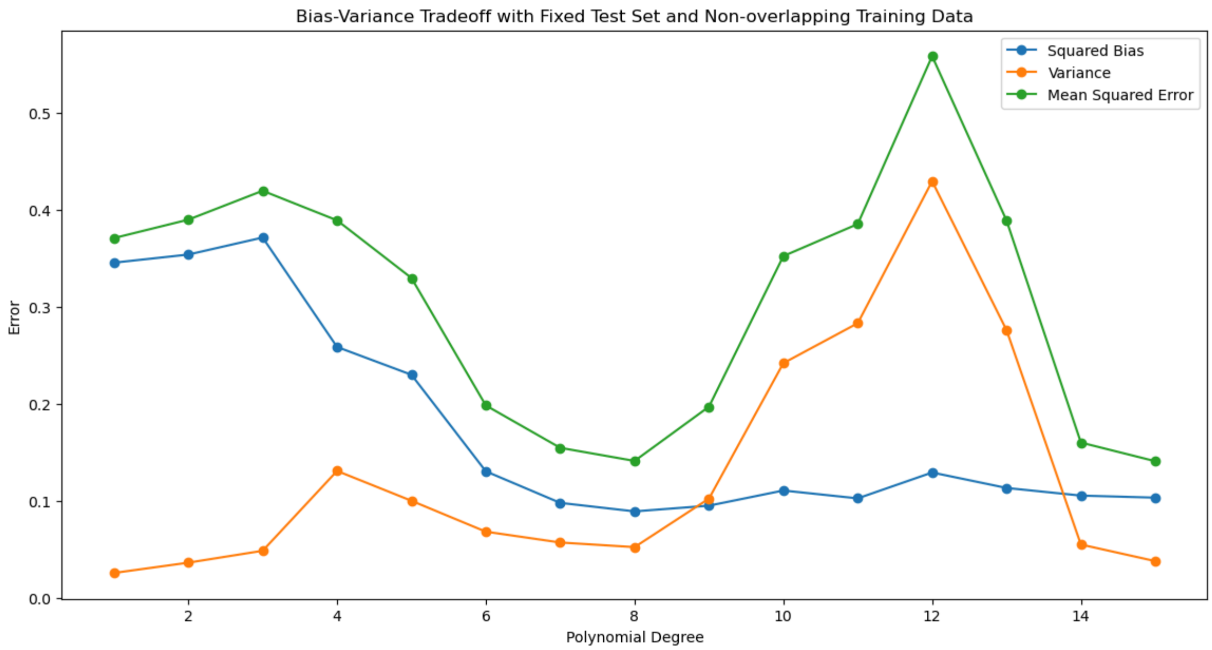


The estimators g_1(x) and g_3(x) are severely underfitting the data, while g_10(x) is overfitting the data.

Part 4: Bias-Variance Tradeoff

In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

# Function to generate dataset excluding test samples
def generate_training_dataset(test_set=None, n_train_samples=40):
    x_all = np.linspace(0, 10, 1000)
    y_all = x_all + np.sin(1.5 * x_all) + np.random.normal(0, 0.3, size=x_al

    if test_set is not None:
        mask = np.isin(x_all, test_set.squeeze(), invert=True)
        x_train_pool = x_all[mask]
        y_train_pool = y_all[mask]
    else:
        x_train_pool = x_all
        y_train_pool = y_all

    # Randomly sample for training
    idx = np.random.choice(len(x_train_pool), n_train_samples, replace=False
    return x_train_pool[idx][:, np.newaxis], y_train_pool[idx]  # Return as

x_test, y_test = generate_training_dataset(None, n_train_samples=10)
np.random.seed(0)
n_datasets = 100
n_samples = 40
degrees = range(1, 16)

# Arrays to store results
mse_test = np.zeros((n_datasets, len(degrees)))
predictions = np.zeros((n_datasets, len(degrees), len(y_test)))


for i in range(n_datasets):
    x_train, y_train = generate_training_dataset(x_test, n_train_samples=n_s
    for j, degree in enumerate(degrees):
        model = make_pipeline(PolynomialFeatures(degree), LinearRegression()
        model.fit(x_train, y_train)
        y_test_pred = model.predict(x_test)

        predictions[i, j, :] = y_test_pred
        mse_test[i, j] = mean_squared_error(y_test, y_test_pred)

# Calculate mean and variance of predictions across datasets
mean_predictions = np.mean(predictions, axis=0)
variance_predictions = np.var(predictions, axis=0)
bias_squared = np.mean((mean_predictions - y_test) ** 2, axis=1)
mse_test_mean = np.mean(mse_test, axis=0)

plt.figure(figsize=(14, 7))
```

```
plt.plot(degrees, bias_squared, label='Squared Bias', marker='o')
plt.plot(degrees, variance_predictions.mean(axis=1), label='Variance', marke
plt.plot(degrees, mse_test_mean, label='Mean Squared Error', marker='o')
plt.xlabel('Polynomial Degree')
plt.ylabel('Error')
plt.title('Bias-Variance Tradeoff with Fixed Test Set and Non-overlapping Tr
plt.legend()
plt.show()
```



The best model is at degree 8.

Part 5: L2 Regularization

In [ ]:

```
import numpy as np
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import Ridge
from sklearn.metrics import mean_squared_error


def generate_training_dataset(test_set=None, n_train_samples=40):
    x_all = np.linspace(0, 10, 1000)  # Generate a large pool of data points
    y_all = x_all + np.sin(1.5 * x_all) + np.random.normal(0, 0.3, size=x_al


    if test_set is not None:
        mask = np.isin(x_all, test_set.squeeze(), invert=True)
        x_train_pool = x_all[mask]  # Exclude test samples from the x pool
        y_train_pool = y_all[mask]
    else:
        x_train_pool = x_all
        y_train_pool = y_all

    # Randomly sample from the remaining points for train
```

```
    idx = np.random.choice(len(x_train_pool), n_train_samples, replace=False
    return x_train_pool[idx][:, np.newaxis], y_train_pool[idx]

#fixed test set (size 10)
x_test, y_test = generate_training_dataset(None, n_train_samples=10)
degree = 10
alpha = 1.0

n_datasets = 100
n_samples = 40

# Arrays to store results
mse_test_ridge = np.zeros(n_datasets)
predictions_ridge = np.zeros((n_datasets, len(y_test)))


for i in range(n_datasets):

    x_train, y_train = generate_training_dataset(x_test, n_train_samples=n_s

    #Ridge regression
    model_ridge = make_pipeline(PolynomialFeatures(degree), Ridge(alpha=alph
    model_ridge.fit(x_train, y_train)
    y_test_pred_ridge = model_ridge.predict(x_test)

    mse_test_ridge[i] = mean_squared_error(y_test, y_test_pred_ridge)
    predictions_ridge[i, :] = y_test_pred_ridge


mean_predictions_ridge = np.mean(predictions_ridge, axis=0)
variance_predictions_ridge = np.var(predictions_ridge, axis=0)
bias_squared_ridge = np.mean((mean_predictions_ridge - y_test) ** 2)
mse_test_mean_ridge = np.mean(mse_test_ridge)


print(f"Unregularized Model (Degree {degree}):")
print(f"  Squared Bias: {bias_squared[degree-1]:.4f}")
print(f"  Variance: {variance_predictions[degree-1].mean():.4f}")
print(f"  Mean Squared Error: {mse_test_mean[degree-1]:.4f}")

print(f"Regularized Model (Degree {degree}, Alpha {alpha}):")
print(f"  Squared Bias: {bias_squared_ridge:.4f}")
print(f"  Variance: {variance_predictions_ridge.mean():.4f}")
print(f"  Mean Squared Error: {mse_test_mean_ridge:.4f}")
```

```
Unregularized Model (Degree 10):
  Squared Bias: 0.1104
  Variance: 0.2418
  Mean Squared Error: 0.3522
Regularized Model (Degree 10, Alpha 1.0):
  Squared Bias: 0.2116
  Variance: 0.0381
  Mean Squared Error: 0.2497
```

```
/Users/bytedance/Documents/anaconda3/envs/COMS6998/lib/python3.9/site-packag
es/sklearn/linear_model/_ridge.py:216: LinAlgWarning: Ill-conditioned matrix
(rcond=5.74457e-21): result may not be accurate.
  return linalg.solve(A, Xy, assume_a="pos", overwrite_a=True).T
/Users/bytedance/Documents/anaconda3/envs/COMS6998/lib/python3.9/site-packag
es/sklearn/linear_model/_ridge.py:216: LinAlgWarning: Ill-conditioned matrix
(rcond=5.11292e-21): result may not be accurate.
  return linalg.solve(A, Xy, assume_a="pos", overwrite_a=True).T
/Users/bytedance/Documents/anaconda3/envs/COMS6998/lib/python3.9/site-packag
es/sklearn/linear_model/_ridge.py:216: LinAlgWarning: Ill-conditioned matrix
(rcond=1.37722e-20): result may not be accurate.
  return linalg.solve(A, Xy, assume_a="pos", overwrite_a=True).T
/Users/bytedance/Documents/anaconda3/envs/COMS6998/lib/python3.9/site-packag
es/sklearn/linear_model/_ridge.py:216: LinAlgWarning: Ill-conditioned matrix
(rcond=5.65029e-21): result may not be accurate.
  return linalg.solve(A, Xy, assume_a="pos", overwrite_a=True).T
/Users/bytedance/Documents/anaconda3/envs/COMS6998/lib/python3.9/site-packag
es/sklearn/linear_model/_ridge.py:216: LinAlgWarning: Ill-conditioned matrix
(rcond=3.14654e-21): result may not be accurate.
  return linalg.solve(A, Xy, assume_a="pos", overwrite_a=True).T
/Users/bytedance/Documents/anaconda3/envs/COMS6998/lib/python3.9/site-packag
es/sklearn/linear_model/_ridge.py:216: LinAlgWarning: Ill-conditioned matrix
(rcond=5.35538e-21): result may not be accurate.
  return linalg.solve(A, Xy, assume_a="pos", overwrite_a=True).T
/Users/bytedance/Documents/anaconda3/envs/COMS6998/lib/python3.9/site-packag
es/sklearn/linear_model/_ridge.py:216: LinAlgWarning: Ill-conditioned matrix
(rcond=2.16829e-21): result may not be accurate.
  return linalg.solve(A, Xy, assume_a="pos", overwrite_a=True).T
/Users/bytedance/Documents/anaconda3/envs/COMS6998/lib/python3.9/site-packag
es/sklearn/linear_model/_ridge.py:216: LinAlgWarning: Ill-conditioned matrix
(rcond=4.84292e-21): result may not be accurate.
  return linalg.solve(A, Xy, assume_a="pos", overwrite_a=True).T
/Users/bytedance/Documents/anaconda3/envs/COMS6998/lib/python3.9/site-packag
es/sklearn/linear_model/_ridge.py:216: LinAlgWarning: Ill-conditioned matrix
(rcond=1.08856e-20): result may not be accurate.
  return linalg.solve(A, Xy, assume_a="pos", overwrite_a=True).T
/Users/bytedance/Documents/anaconda3/envs/COMS6998/lib/python3.9/site-packag
es/sklearn/linear_model/_ridge.py:216: LinAlgWarning: Ill-conditioned matrix
(rcond=5.5261e-21): result may not be accurate.
  return linalg.solve(A, Xy, assume_a="pos", overwrite_a=True).T
/Users/bytedance/Documents/anaconda3/envs/COMS6998/lib/python3.9/site-packag
es/sklearn/linear_model/_ridge.py:216: LinAlgWarning: Ill-conditioned matrix
(rcond=5.91355e-21): result may not be accurate.
  return linalg.solve(A, Xy, assume_a="pos", overwrite_a=True).T
/Users/bytedance/Documents/anaconda3/envs/COMS6998/lib/python3.9/site-packag
es/sklearn/linear_model/_ridge.py:216: LinAlgWarning: Ill-conditioned matrix
(rcond=5.8503e-21): result may not be accurate.
  return linalg.solve(A, Xy, assume_a="pos", overwrite_a=True).T
/Users/bytedance/Documents/anaconda3/envs/COMS6998/lib/python3.9/site-packag
es/sklearn/linear_model/_ridge.py:216: LinAlgWarning: Ill-conditioned matrix
(rcond=7.10461e-21): result may not be accurate.
  return linalg.solve(A, Xy, assume_a="pos", overwrite_a=True).T
/Users/bytedance/Documents/anaconda3/envs/COMS6998/lib/python3.9/site-packag
es/sklearn/linear_model/_ridge.py:216: LinAlgWarning: Ill-conditioned matrix
(rcond=7.96422e-21): result may not be accurate.
  return linalg.solve(A, Xy, assume_a="pos", overwrite_a=True).T
```

```
/Users/bytedance/Documents/anaconda3/envs/COMS6998/lib/python3.9/site-packag
es/sklearn/linear_model/_ridge.py:216: LinAlgWarning: Ill-conditioned matrix
(rcond=2.8228e-21): result may not be accurate.
  return linalg.solve(A, Xy, assume_a="pos", overwrite_a=True).T
/Users/bytedance/Documents/anaconda3/envs/COMS6998/lib/python3.9/site-packag
es/sklearn/linear_model/_ridge.py:216: LinAlgWarning: Ill-conditioned matrix
(rcond=1.21069e-20): result may not be accurate.
  return linalg.solve(A, Xy, assume_a="pos", overwrite_a=True).T
/Users/bytedance/Documents/anaconda3/envs/COMS6998/lib/python3.9/site-packag
es/sklearn/linear_model/_ridge.py:216: LinAlgWarning: Ill-conditioned matrix
(rcond=3.72526e-21): result may not be accurate.
  return linalg.solve(A, Xy, assume_a="pos", overwrite_a=True).T
/Users/bytedance/Documents/anaconda3/envs/COMS6998/lib/python3.9/site-packag
es/sklearn/linear_model/_ridge.py:216: LinAlgWarning: Ill-conditioned matrix
(rcond=4.89184e-21): result may not be accurate.
  return linalg.solve(A, Xy, assume_a="pos", overwrite_a=True).T
/Users/bytedance/Documents/anaconda3/envs/COMS6998/lib/python3.9/site-packag
es/sklearn/linear_model/_ridge.py:216: LinAlgWarning: Ill-conditioned matrix
(rcond=3.58541e-21): result may not be accurate.
  return linalg.solve(A, Xy, assume_a="pos", overwrite_a=True).T
/Users/bytedance/Documents/anaconda3/envs/COMS6998/lib/python3.9/site-packag
es/sklearn/linear_model/_ridge.py:216: LinAlgWarning: Ill-conditioned matrix
(rcond=1.98561e-20): result may not be accurate.
  return linalg.solve(A, Xy, assume_a="pos", overwrite_a=True).T
/Users/bytedance/Documents/anaconda3/envs/COMS6998/lib/python3.9/site-packag
es/sklearn/linear_model/_ridge.py:216: LinAlgWarning: Ill-conditioned matrix
(rcond=4.2683e-21): result may not be accurate.
  return linalg.solve(A, Xy, assume_a="pos", overwrite_a=True).T
/Users/bytedance/Documents/anaconda3/envs/COMS6998/lib/python3.9/site-packag
es/sklearn/linear_model/_ridge.py:216: LinAlgWarning: Ill-conditioned matrix
(rcond=2.5423e-21): result may not be accurate.
  return linalg.solve(A, Xy, assume_a="pos", overwrite_a=True).T
/Users/bytedance/Documents/anaconda3/envs/COMS6998/lib/python3.9/site-packag
es/sklearn/linear_model/_ridge.py:216: LinAlgWarning: Ill-conditioned matrix
(rcond=4.67567e-21): result may not be accurate.
  return linalg.solve(A, Xy, assume_a="pos", overwrite_a=True).T
/Users/bytedance/Documents/anaconda3/envs/COMS6998/lib/python3.9/site-packag
es/sklearn/linear_model/_ridge.py:216: LinAlgWarning: Ill-conditioned matrix
(rcond=3.27761e-21): result may not be accurate.
  return linalg.solve(A, Xy, assume_a="pos", overwrite_a=True).T
/Users/bytedance/Documents/anaconda3/envs/COMS6998/lib/python3.9/site-packag
es/sklearn/linear_model/_ridge.py:216: LinAlgWarning: Ill-conditioned matrix
(rcond=4.51958e-21): result may not be accurate.
  return linalg.solve(A, Xy, assume_a="pos", overwrite_a=True).T
/Users/bytedance/Documents/anaconda3/envs/COMS6998/lib/python3.9/site-packag
es/sklearn/linear_model/_ridge.py:216: LinAlgWarning: Ill-conditioned matrix
(rcond=6.14133e-21): result may not be accurate.
  return linalg.solve(A, Xy, assume_a="pos", overwrite_a=True).T
/Users/bytedance/Documents/anaconda3/envs/COMS6998/lib/python3.9/site-packag
es/sklearn/linear_model/_ridge.py:216: LinAlgWarning: Ill-conditioned matrix
(rcond=4.02724e-21): result may not be accurate.
  return linalg.solve(A, Xy, assume_a="pos", overwrite_a=True).T
/Users/bytedance/Documents/anaconda3/envs/COMS6998/lib/python3.9/site-packag
es/sklearn/linear_model/_ridge.py:216: LinAlgWarning: Ill-conditioned matrix
(rcond=8.08975e-21): result may not be accurate.
  return linalg.solve(A, Xy, assume_a="pos", overwrite_a=True).T
```

```
/Users/bytedance/Documents/anaconda3/envs/COMS6998/lib/python3.9/site-packag
es/sklearn/linear_model/_ridge.py:216: LinAlgWarning: Ill-conditioned matrix
(rcond=1.5753e-20): result may not be accurate.
  return linalg.solve(A, Xy, assume_a="pos", overwrite_a=True).T
/Users/bytedance/Documents/anaconda3/envs/COMS6998/lib/python3.9/site-packag
es/sklearn/linear_model/_ridge.py:216: LinAlgWarning: Ill-conditioned matrix
(rcond=3.54163e-21): result may not be accurate.
  return linalg.solve(A, Xy, assume_a="pos", overwrite_a=True).T
/Users/bytedance/Documents/anaconda3/envs/COMS6998/lib/python3.9/site-packag
es/sklearn/linear_model/_ridge.py:216: LinAlgWarning: Ill-conditioned matrix
(rcond=4.01958e-21): result may not be accurate.
  return linalg.solve(A, Xy, assume_a="pos", overwrite_a=True).T
/Users/bytedance/Documents/anaconda3/envs/COMS6998/lib/python3.9/site-packag
es/sklearn/linear_model/_ridge.py:216: LinAlgWarning: Ill-conditioned matrix
(rcond=8.40025e-21): result may not be accurate.
  return linalg.solve(A, Xy, assume_a="pos", overwrite_a=True).T
/Users/bytedance/Documents/anaconda3/envs/COMS6998/lib/python3.9/site-packag
es/sklearn/linear_model/_ridge.py:216: LinAlgWarning: Ill-conditioned matrix
(rcond=9.41755e-21): result may not be accurate.
  return linalg.solve(A, Xy, assume_a="pos", overwrite_a=True).T
/Users/bytedance/Documents/anaconda3/envs/COMS6998/lib/python3.9/site-packag
es/sklearn/linear_model/_ridge.py:216: LinAlgWarning: Ill-conditioned matrix
(rcond=3.1765e-21): result may not be accurate.
  return linalg.solve(A, Xy, assume_a="pos", overwrite_a=True).T
/Users/bytedance/Documents/anaconda3/envs/COMS6998/lib/python3.9/site-packag
es/sklearn/linear_model/_ridge.py:216: LinAlgWarning: Ill-conditioned matrix
(rcond=4.56372e-21): result may not be accurate.
  return linalg.solve(A, Xy, assume_a="pos", overwrite_a=True).T
/Users/bytedance/Documents/anaconda3/envs/COMS6998/lib/python3.9/site-packag
es/sklearn/linear_model/_ridge.py:216: LinAlgWarning: Ill-conditioned matrix
(rcond=6.19589e-21): result may not be accurate.
  return linalg.solve(A, Xy, assume_a="pos", overwrite_a=True).T
/Users/bytedance/Documents/anaconda3/envs/COMS6998/lib/python3.9/site-packag
es/sklearn/linear_model/_ridge.py:216: LinAlgWarning: Ill-conditioned matrix
(rcond=3.8962e-21): result may not be accurate.
  return linalg.solve(A, Xy, assume_a="pos", overwrite_a=True).T
/Users/bytedance/Documents/anaconda3/envs/COMS6998/lib/python3.9/site-packag
es/sklearn/linear_model/_ridge.py:216: LinAlgWarning: Ill-conditioned matrix
(rcond=3.8271e-21): result may not be accurate.
  return linalg.solve(A, Xy, assume_a="pos", overwrite_a=True).T
/Users/bytedance/Documents/anaconda3/envs/COMS6998/lib/python3.9/site-packag
es/sklearn/linear_model/_ridge.py:216: LinAlgWarning: Ill-conditioned matrix
(rcond=1.64109e-20): result may not be accurate.
  return linalg.solve(A, Xy, assume_a="pos", overwrite_a=True).T
/Users/bytedance/Documents/anaconda3/envs/COMS6998/lib/python3.9/site-packag
es/sklearn/linear_model/_ridge.py:216: LinAlgWarning: Ill-conditioned matrix
(rcond=2.20173e-21): result may not be accurate.
  return linalg.solve(A, Xy, assume_a="pos", overwrite_a=True).T
/Users/bytedance/Documents/anaconda3/envs/COMS6998/lib/python3.9/site-packag
es/sklearn/linear_model/_ridge.py:216: LinAlgWarning: Ill-conditioned matrix
(rcond=2.44277e-21): result may not be accurate.
  return linalg.solve(A, Xy, assume_a="pos", overwrite_a=True).T
/Users/bytedance/Documents/anaconda3/envs/COMS6998/lib/python3.9/site-packag
es/sklearn/linear_model/_ridge.py:216: LinAlgWarning: Ill-conditioned matrix
(rcond=2.80385e-20): result may not be accurate.
  return linalg.solve(A, Xy, assume_a="pos", overwrite_a=True).T
```

```
/Users/bytedance/Documents/anaconda3/envs/COMS6998/lib/python3.9/site-packag
es/sklearn/linear_model/_ridge.py:216: LinAlgWarning: Ill-conditioned matrix
(rcond=6.88431e-21): result may not be accurate.
  return linalg.solve(A, Xy, assume_a="pos", overwrite_a=True).T
/Users/bytedance/Documents/anaconda3/envs/COMS6998/lib/python3.9/site-packag
es/sklearn/linear_model/_ridge.py:216: LinAlgWarning: Ill-conditioned matrix
(rcond=8.1635e-21): result may not be accurate.
  return linalg.solve(A, Xy, assume_a="pos", overwrite_a=True).T
/Users/bytedance/Documents/anaconda3/envs/COMS6998/lib/python3.9/site-packag
es/sklearn/linear_model/_ridge.py:216: LinAlgWarning: Ill-conditioned matrix
(rcond=3.2025e-21): result may not be accurate.
  return linalg.solve(A, Xy, assume_a="pos", overwrite_a=True).T
/Users/bytedance/Documents/anaconda3/envs/COMS6998/lib/python3.9/site-packag
es/sklearn/linear_model/_ridge.py:216: LinAlgWarning: Ill-conditioned matrix
(rcond=2.66808e-21): result may not be accurate.
  return linalg.solve(A, Xy, assume_a="pos", overwrite_a=True).T
/Users/bytedance/Documents/anaconda3/envs/COMS6998/lib/python3.9/site-packag
es/sklearn/linear_model/_ridge.py:216: LinAlgWarning: Ill-conditioned matrix
(rcond=1.01068e-20): result may not be accurate.
  return linalg.solve(A, Xy, assume_a="pos", overwrite_a=True).T
/Users/bytedance/Documents/anaconda3/envs/COMS6998/lib/python3.9/site-packag
es/sklearn/linear_model/_ridge.py:216: LinAlgWarning: Ill-conditioned matrix
(rcond=6.44486e-21): result may not be accurate.
  return linalg.solve(A, Xy, assume_a="pos", overwrite_a=True).T
/Users/bytedance/Documents/anaconda3/envs/COMS6998/lib/python3.9/site-packag
es/sklearn/linear_model/_ridge.py:216: LinAlgWarning: Ill-conditioned matrix
(rcond=3.52326e-21): result may not be accurate.
  return linalg.solve(A, Xy, assume_a="pos", overwrite_a=True).T
/Users/bytedance/Documents/anaconda3/envs/COMS6998/lib/python3.9/site-packag
es/sklearn/linear_model/_ridge.py:216: LinAlgWarning: Ill-conditioned matrix
(rcond=1.71974e-20): result may not be accurate.
  return linalg.solve(A, Xy, assume_a="pos", overwrite_a=True).T
/Users/bytedance/Documents/anaconda3/envs/COMS6998/lib/python3.9/site-packag
es/sklearn/linear_model/_ridge.py:216: LinAlgWarning: Ill-conditioned matrix
(rcond=5.15387e-21): result may not be accurate.
  return linalg.solve(A, Xy, assume_a="pos", overwrite_a=True).T
/Users/bytedance/Documents/anaconda3/envs/COMS6998/lib/python3.9/site-packag
es/sklearn/linear_model/_ridge.py:216: LinAlgWarning: Ill-conditioned matrix
(rcond=8.87974e-21): result may not be accurate.
  return linalg.solve(A, Xy, assume_a="pos", overwrite_a=True).T
/Users/bytedance/Documents/anaconda3/envs/COMS6998/lib/python3.9/site-packag
es/sklearn/linear_model/_ridge.py:216: LinAlgWarning: Ill-conditioned matrix
(rcond=5.4305e-21): result may not be accurate.
  return linalg.solve(A, Xy, assume_a="pos", overwrite_a=True).T
/Users/bytedance/Documents/anaconda3/envs/COMS6998/lib/python3.9/site-packag
es/sklearn/linear_model/_ridge.py:216: LinAlgWarning: Ill-conditioned matrix
(rcond=7.14242e-21): result may not be accurate.
  return linalg.solve(A, Xy, assume_a="pos", overwrite_a=True).T
/Users/bytedance/Documents/anaconda3/envs/COMS6998/lib/python3.9/site-packag
es/sklearn/linear_model/_ridge.py:216: LinAlgWarning: Ill-conditioned matrix
(rcond=2.72211e-21): result may not be accurate.
  return linalg.solve(A, Xy, assume_a="pos", overwrite_a=True).T
/Users/bytedance/Documents/anaconda3/envs/COMS6998/lib/python3.9/site-packag
es/sklearn/linear_model/_ridge.py:216: LinAlgWarning: Ill-conditioned matrix
(rcond=1.09692e-20): result may not be accurate.
  return linalg.solve(A, Xy, assume_a="pos", overwrite_a=True).T
```

```
/Users/bytedance/Documents/anaconda3/envs/COMS6998/lib/python3.9/site-packag
es/sklearn/linear_model/_ridge.py:216: LinAlgWarning: Ill-conditioned matrix
(rcond=5.90181e-21): result may not be accurate.
  return linalg.solve(A, Xy, assume_a="pos", overwrite_a=True).T
/Users/bytedance/Documents/anaconda3/envs/COMS6998/lib/python3.9/site-packag
es/sklearn/linear_model/_ridge.py:216: LinAlgWarning: Ill-conditioned matrix
(rcond=2.48205e-21): result may not be accurate.
  return linalg.solve(A, Xy, assume_a="pos", overwrite_a=True).T
/Users/bytedance/Documents/anaconda3/envs/COMS6998/lib/python3.9/site-packag
es/sklearn/linear_model/_ridge.py:216: LinAlgWarning: Ill-conditioned matrix
(rcond=3.15675e-21): result may not be accurate.
  return linalg.solve(A, Xy, assume_a="pos", overwrite_a=True).T
/Users/bytedance/Documents/anaconda3/envs/COMS6998/lib/python3.9/site-packag
es/sklearn/linear_model/_ridge.py:216: LinAlgWarning: Ill-conditioned matrix
(rcond=5.43158e-21): result may not be accurate.
  return linalg.solve(A, Xy, assume_a="pos", overwrite_a=True).T
/Users/bytedance/Documents/anaconda3/envs/COMS6998/lib/python3.9/site-packag
es/sklearn/linear_model/_ridge.py:216: LinAlgWarning: Ill-conditioned matrix
(rcond=4.67297e-21): result may not be accurate.
  return linalg.solve(A, Xy, assume_a="pos", overwrite_a=True).T
/Users/bytedance/Documents/anaconda3/envs/COMS6998/lib/python3.9/site-packag
es/sklearn/linear_model/_ridge.py:216: LinAlgWarning: Ill-conditioned matrix
(rcond=5.20322e-21): result may not be accurate.
  return linalg.solve(A, Xy, assume_a="pos", overwrite_a=True).T
/Users/bytedance/Documents/anaconda3/envs/COMS6998/lib/python3.9/site-packag
es/sklearn/linear_model/_ridge.py:216: LinAlgWarning: Ill-conditioned matrix
(rcond=1.24456e-20): result may not be accurate.
  return linalg.solve(A, Xy, assume_a="pos", overwrite_a=True).T
/Users/bytedance/Documents/anaconda3/envs/COMS6998/lib/python3.9/site-packag
es/sklearn/linear_model/_ridge.py:216: LinAlgWarning: Ill-conditioned matrix
(rcond=8.96587e-21): result may not be accurate.
  return linalg.solve(A, Xy, assume_a="pos", overwrite_a=True).T
/Users/bytedance/Documents/anaconda3/envs/COMS6998/lib/python3.9/site-packag
es/sklearn/linear_model/_ridge.py:216: LinAlgWarning: Ill-conditioned matrix
(rcond=5.0855e-21): result may not be accurate.
  return linalg.solve(A, Xy, assume_a="pos", overwrite_a=True).T
/Users/bytedance/Documents/anaconda3/envs/COMS6998/lib/python3.9/site-packag
es/sklearn/linear_model/_ridge.py:216: LinAlgWarning: Ill-conditioned matrix
(rcond=3.15014e-21): result may not be accurate.
  return linalg.solve(A, Xy, assume_a="pos", overwrite_a=True).T
/Users/bytedance/Documents/anaconda3/envs/COMS6998/lib/python3.9/site-packag
es/sklearn/linear_model/_ridge.py:216: LinAlgWarning: Ill-conditioned matrix
(rcond=8.81863e-21): result may not be accurate.
  return linalg.solve(A, Xy, assume_a="pos", overwrite_a=True).T
/Users/bytedance/Documents/anaconda3/envs/COMS6998/lib/python3.9/site-packag
es/sklearn/linear_model/_ridge.py:216: LinAlgWarning: Ill-conditioned matrix
(rcond=6.32581e-21): result may not be accurate.
  return linalg.solve(A, Xy, assume_a="pos", overwrite_a=True).T
/Users/bytedance/Documents/anaconda3/envs/COMS6998/lib/python3.9/site-packag
es/sklearn/linear_model/_ridge.py:216: LinAlgWarning: Ill-conditioned matrix
(rcond=3.42182e-21): result may not be accurate.
  return linalg.solve(A, Xy, assume_a="pos", overwrite_a=True).T
/Users/bytedance/Documents/anaconda3/envs/COMS6998/lib/python3.9/site-packag
es/sklearn/linear_model/_ridge.py:216: LinAlgWarning: Ill-conditioned matrix
(rcond=8.73332e-21): result may not be accurate.
  return linalg.solve(A, Xy, assume_a="pos", overwrite_a=True).T
```

```
/Users/bytedance/Documents/anaconda3/envs/COMS6998/lib/python3.9/site-packag
es/sklearn/linear_model/_ridge.py:216: LinAlgWarning: Ill-conditioned matrix
(rcond=4.03142e-21): result may not be accurate.
  return linalg.solve(A, Xy, assume_a="pos", overwrite_a=True).T
/Users/bytedance/Documents/anaconda3/envs/COMS6998/lib/python3.9/site-packag
es/sklearn/linear_model/_ridge.py:216: LinAlgWarning: Ill-conditioned matrix
(rcond=8.25931e-21): result may not be accurate.
  return linalg.solve(A, Xy, assume_a="pos", overwrite_a=True).T
/Users/bytedance/Documents/anaconda3/envs/COMS6998/lib/python3.9/site-packag
es/sklearn/linear_model/_ridge.py:216: LinAlgWarning: Ill-conditioned matrix
(rcond=6.70856e-21): result may not be accurate.
  return linalg.solve(A, Xy, assume_a="pos", overwrite_a=True).T
/Users/bytedance/Documents/anaconda3/envs/COMS6998/lib/python3.9/site-packag
es/sklearn/linear_model/_ridge.py:216: LinAlgWarning: Ill-conditioned matrix
(rcond=3.62506e-21): result may not be accurate.
  return linalg.solve(A, Xy, assume_a="pos", overwrite_a=True).T
/Users/bytedance/Documents/anaconda3/envs/COMS6998/lib/python3.9/site-packag
es/sklearn/linear_model/_ridge.py:216: LinAlgWarning: Ill-conditioned matrix
(rcond=5.19823e-21): result may not be accurate.
  return linalg.solve(A, Xy, assume_a="pos", overwrite_a=True).T
/Users/bytedance/Documents/anaconda3/envs/COMS6998/lib/python3.9/site-packag
es/sklearn/linear_model/_ridge.py:216: LinAlgWarning: Ill-conditioned matrix
(rcond=5.58836e-21): result may not be accurate.
  return linalg.solve(A, Xy, assume_a="pos", overwrite_a=True).T
/Users/bytedance/Documents/anaconda3/envs/COMS6998/lib/python3.9/site-packag
es/sklearn/linear_model/_ridge.py:216: LinAlgWarning: Ill-conditioned matrix
(rcond=1.38099e-20): result may not be accurate.
  return linalg.solve(A, Xy, assume_a="pos", overwrite_a=True).T
/Users/bytedance/Documents/anaconda3/envs/COMS6998/lib/python3.9/site-packag
es/sklearn/linear_model/_ridge.py:216: LinAlgWarning: Ill-conditioned matrix
(rcond=2.07996e-21): result may not be accurate.
  return linalg.solve(A, Xy, assume_a="pos", overwrite_a=True).T
/Users/bytedance/Documents/anaconda3/envs/COMS6998/lib/python3.9/site-packag
es/sklearn/linear_model/_ridge.py:216: LinAlgWarning: Ill-conditioned matrix
(rcond=3.6614e-21): result may not be accurate.
  return linalg.solve(A, Xy, assume_a="pos", overwrite_a=True).T
/Users/bytedance/Documents/anaconda3/envs/COMS6998/lib/python3.9/site-packag
es/sklearn/linear_model/_ridge.py:216: LinAlgWarning: Ill-conditioned matrix
(rcond=2.10873e-20): result may not be accurate.
  return linalg.solve(A, Xy, assume_a="pos", overwrite_a=True).T
/Users/bytedance/Documents/anaconda3/envs/COMS6998/lib/python3.9/site-packag
es/sklearn/linear_model/_ridge.py:216: LinAlgWarning: Ill-conditioned matrix
(rcond=4.22142e-21): result may not be accurate.
  return linalg.solve(A, Xy, assume_a="pos", overwrite_a=True).T
/Users/bytedance/Documents/anaconda3/envs/COMS6998/lib/python3.9/site-packag
es/sklearn/linear_model/_ridge.py:216: LinAlgWarning: Ill-conditioned matrix
(rcond=4.09759e-21): result may not be accurate.
  return linalg.solve(A, Xy, assume_a="pos", overwrite_a=True).T
/Users/bytedance/Documents/anaconda3/envs/COMS6998/lib/python3.9/site-packag
es/sklearn/linear_model/_ridge.py:216: LinAlgWarning: Ill-conditioned matrix
(rcond=2.32638e-20): result may not be accurate.
  return linalg.solve(A, Xy, assume_a="pos", overwrite_a=True).T
/Users/bytedance/Documents/anaconda3/envs/COMS6998/lib/python3.9/site-packag
es/sklearn/linear_model/_ridge.py:216: LinAlgWarning: Ill-conditioned matrix
(rcond=5.89028e-21): result may not be accurate.
  return linalg.solve(A, Xy, assume_a="pos", overwrite_a=True).T
```

```
/Users/bytedance/Documents/anaconda3/envs/COMS6998/lib/python3.9/site-packag
es/sklearn/linear_model/_ridge.py:216: LinAlgWarning: Ill-conditioned matrix
(rcond=3.11701e-21): result may not be accurate.
  return linalg.solve(A, Xy, assume_a="pos", overwrite_a=True).T
/Users/bytedance/Documents/anaconda3/envs/COMS6998/lib/python3.9/site-packag
es/sklearn/linear_model/_ridge.py:216: LinAlgWarning: Ill-conditioned matrix
(rcond=4.47675e-21): result may not be accurate.
  return linalg.solve(A, Xy, assume_a="pos", overwrite_a=True).T
/Users/bytedance/Documents/anaconda3/envs/COMS6998/lib/python3.9/site-packag
es/sklearn/linear_model/_ridge.py:216: LinAlgWarning: Ill-conditioned matrix
(rcond=4.52452e-21): result may not be accurate.
  return linalg.solve(A, Xy, assume_a="pos", overwrite_a=True).T
/Users/bytedance/Documents/anaconda3/envs/COMS6998/lib/python3.9/site-packag
es/sklearn/linear_model/_ridge.py:216: LinAlgWarning: Ill-conditioned matrix
(rcond=6.23332e-21): result may not be accurate.
  return linalg.solve(A, Xy, assume_a="pos", overwrite_a=True).T
/Users/bytedance/Documents/anaconda3/envs/COMS6998/lib/python3.9/site-packag
es/sklearn/linear_model/_ridge.py:216: LinAlgWarning: Ill-conditioned matrix
(rcond=5.07605e-21): result may not be accurate.
  return linalg.solve(A, Xy, assume_a="pos", overwrite_a=True).T
/Users/bytedance/Documents/anaconda3/envs/COMS6998/lib/python3.9/site-packag
es/sklearn/linear_model/_ridge.py:216: LinAlgWarning: Ill-conditioned matrix
(rcond=7.6816e-21): result may not be accurate.
  return linalg.solve(A, Xy, assume_a="pos", overwrite_a=True).T
/Users/bytedance/Documents/anaconda3/envs/COMS6998/lib/python3.9/site-packag
es/sklearn/linear_model/_ridge.py:216: LinAlgWarning: Ill-conditioned matrix
(rcond=3.58546e-21): result may not be accurate.
  return linalg.solve(A, Xy, assume_a="pos", overwrite_a=True).T
/Users/bytedance/Documents/anaconda3/envs/COMS6998/lib/python3.9/site-packag
es/sklearn/linear_model/_ridge.py:216: LinAlgWarning: Ill-conditioned matrix
(rcond=5.24255e-21): result may not be accurate.
  return linalg.solve(A, Xy, assume_a="pos", overwrite_a=True).T
/Users/bytedance/Documents/anaconda3/envs/COMS6998/lib/python3.9/site-packag
es/sklearn/linear_model/_ridge.py:216: LinAlgWarning: Ill-conditioned matrix
(rcond=1.07676e-20): result may not be accurate.
  return linalg.solve(A, Xy, assume_a="pos", overwrite_a=True).T
/Users/bytedance/Documents/anaconda3/envs/COMS6998/lib/python3.9/site-packag
es/sklearn/linear_model/_ridge.py:216: LinAlgWarning: Ill-conditioned matrix
(rcond=3.47348e-21): result may not be accurate.
  return linalg.solve(A, Xy, assume_a="pos", overwrite_a=True).T
/Users/bytedance/Documents/anaconda3/envs/COMS6998/lib/python3.9/site-packag
es/sklearn/linear_model/_ridge.py:216: LinAlgWarning: Ill-conditioned matrix
(rcond=6.42758e-21): result may not be accurate.
  return linalg.solve(A, Xy, assume_a="pos", overwrite_a=True).T
/Users/bytedance/Documents/anaconda3/envs/COMS6998/lib/python3.9/site-packag
es/sklearn/linear_model/_ridge.py:216: LinAlgWarning: Ill-conditioned matrix
(rcond=4.0625e-21): result may not be accurate.
  return linalg.solve(A, Xy, assume_a="pos", overwrite_a=True).T
/Users/bytedance/Documents/anaconda3/envs/COMS6998/lib/python3.9/site-packag
es/sklearn/linear_model/_ridge.py:216: LinAlgWarning: Ill-conditioned matrix
(rcond=2.54347e-21): result may not be accurate.
  return linalg.solve(A, Xy, assume_a="pos", overwrite_a=True).T
/Users/bytedance/Documents/anaconda3/envs/COMS6998/lib/python3.9/site-packag
es/sklearn/linear_model/_ridge.py:216: LinAlgWarning: Ill-conditioned matrix
(rcond=3.52508e-21): result may not be accurate.
  return linalg.solve(A, Xy, assume_a="pos", overwrite_a=True).T
```

```
/Users/bytedance/Documents/anaconda3/envs/COMS6998/lib/python3.9/site-packag
es/sklearn/linear_model/_ridge.py:216: LinAlgWarning: Ill-conditioned matrix
(rcond=4.74343e-21): result may not be accurate.
  return linalg.solve(A, Xy, assume_a="pos", overwrite_a=True).T
/Users/bytedance/Documents/anaconda3/envs/COMS6998/lib/python3.9/site-packag
es/sklearn/linear_model/_ridge.py:216: LinAlgWarning: Ill-conditioned matrix
(rcond=2.88995e-21): result may not be accurate.
  return linalg.solve(A, Xy, assume_a="pos", overwrite_a=True).T
```

Conclusion

Bias:

- **Unregularized Model:** Squared bias is 0.1104.
- **Regularized Model:** Squared bias is 0.2116.
- **Ans:** The regularized model has a higher bias, as expected, due to reduced model flexibility.

Variance:

- **Unregularized Model:** Variance is 0.2418.
- **Regularized Model:** Variance is 0.0381.
- **Ans:** The regularized model has much lower variance, which is expected as regularization reduces sensitivity to training data.

Mean Squared Error (MSE):

- **Unregularized Model:** MSE is 0.3522.
- **Regularized Model:** MSE is 0.2497.
- **Ans:** The regularized model has a lower MSE, indicating better overall performance.

Answer: The regularized model has higher bias, but achieves much lower variance and MSE. This shows better generalization and makes it the better model.

Problem2.1

- **ROC Curve:** TPR/FPR. True negatives (TN) matter because the false positive rate (FPR) depends on TN:

$$\text{FPR} = \frac{\text{FP}}{\text{FP} + \text{TN}}$$

The ROC curve plots the true positive rate (TPR) against FPR, so TN is essential for computing FPR.

- **PR Curve:** True negatives do not matter. The PR curve uses precision and recall, which depend on true positives (TP), false positives (FP), and false negatives (FN):

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

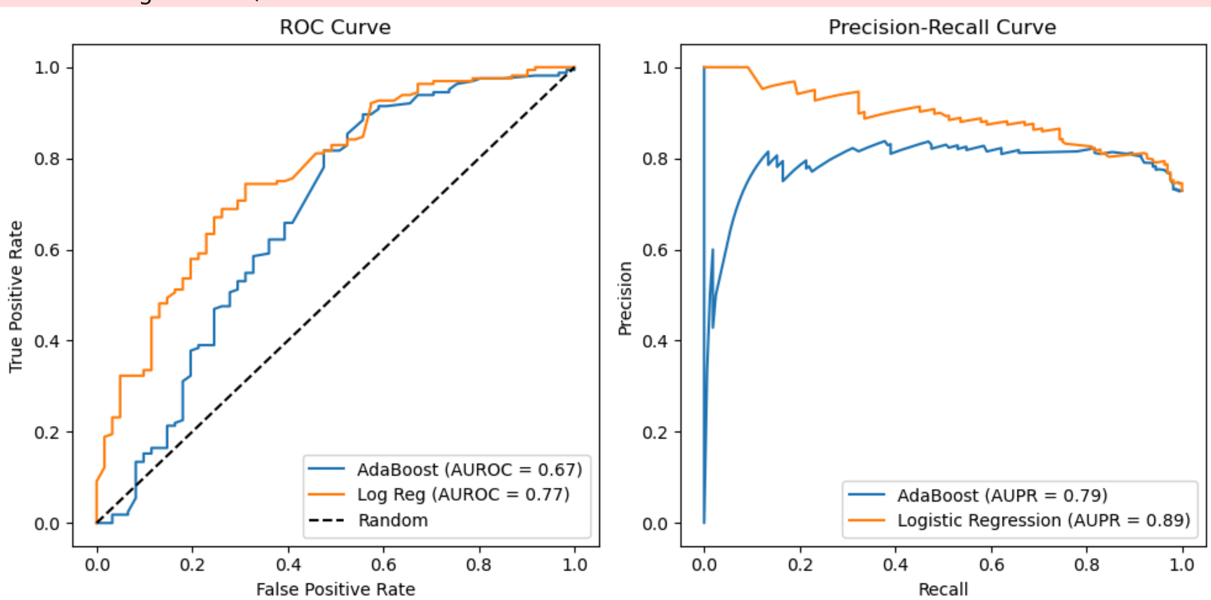
$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

Each point on the ROC curve corresponds to a unique point on the PR curve because both curves are based on the same TP, FP, FN, TN. For any decision threshold, the counts of TP, FP, FN, and TN are fixed, producing both an ROC and a PR point. So while ROC curves are used for balanced data, PR curves are used for imbalanced data, however their metrics are based on the same counts of TP, FP, FN, TN.

In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import fetch_openml
from sklearn.model_selection import train_test_split
from sklearn.ensemble import AdaBoostClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_curve, precision_recall_curve, auc

X, y = fetch_openml(name='blood-transfusion-service-center', version=1, as_f
y = y.astype(int).replace({2: 0})

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, rar
ada = AdaBoostClassifier(n_estimators=100, random_state=0)
logreg = LogisticRegression(solver='liblinear')

ada.fit(X_train, y_train)
logreg.fit(X_train, y_train)


ada_probs = ada.predict_proba(X_test)[:, 1]
logreg_probs = logreg.predict_proba(X_test)[:, 1]

# Generate ROC/ PR curves
fpr_ada, tpr_ada, _ = roc_curve(y_test, ada_probs)
```

```
fpr_logreg, tpr_logreg, _ = roc_curve(y_test, logreg_probs)

precision_ada, recall_ada, _ = precision_recall_curve(y_test, ada_probs)
precision_logreg, recall_logreg, _ = precision_recall_curve(y_test, logreg_p

plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.plot(fpr_ada, tpr_ada, label=f'AdaBoost (AUROC = {auc(fpr_ada, tpr_ada):
plt.plot(fpr_logreg, tpr_logreg, label=f'Log Reg (AUROC = {auc(fpr_logreg, t
plt.plot([0, 1], [0, 1], 'k--', label='Random')
plt.title('ROC Curve')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(recall_ada, precision_ada, label=f'AdaBoost (AUPR = {auc(recall_ada
plt.plot(recall_logreg, precision_logreg, label=f'Logistic Regression (AUPR
plt.title('Precision-Recall Curve')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.legend()


plt.tight_layout()
plt.show()

all_pos = sum(y_test == 1) / len(y_test)
print(f"All-positive classifier ROC point: (1.0, {all_pos})")
print(f"All-positive classifier PR point: Precision = {all_pos}, Recall = 1.
```

```
/Users/bytedance/Documents/anaconda3/envs/COMS6998/lib/python3.9/site-packag
es/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R algo
rithm (the default) is deprecated and will be removed in 1.6. Use the SAMME
algorithm to circumvent this warning.
  warnings.warn(
```



```
All-positive classifier ROC point: (1.0, 0.7288888888888889)
All-positive classifier PR point: Precision = 0.7288888888888889, Recall =
1.0
```

In [ ]:
```
def prg_curve(precision, recall, pi, epsilon=1e-10):
    precision_gain = np.where(precision < 1, (precision - pi) / (pi * (1 - p
    recall_gain = np.where(recall < 1, (recall - pi) / (pi * (1 - recall + e
    return precision_gain, recall_gain

precision_gain_ada, recall_gain_ada = prg_curve(precision_ada, recall_ada, p

precision_gain_logreg, recall_gain_logreg = prg_curve(precision_logreg, reca


sorted_indices_ada = np.argsort(recall_gain_ada)
recall_gain_ada_sorted = recall_gain_ada[sorted_indices_ada]
precision_gain_ada_sorted = precision_gain_ada[sorted_indices_ada]


sorted_indices_logreg = np.argsort(recall_gain_logreg)
recall_gain_logreg_sorted = recall_gain_logreg[sorted_indices_logreg]
precision_gain_logreg_sorted = precision_gain_logreg[sorted_indices_logreg]

auprg_ada = auc(recall_gain_ada_sorted, precision_gain_ada_sorted)
auprg_logreg = auc(recall_gain_logreg_sorted, precision_gain_logreg_sorted)

print(f"AdaBoost - AUROC: {auroc_ada:.3f}, AUPR: {aupr_ada:.3f}, AUPRG: {aup
print(f"Logistic Regression - AUROC: {auroc_logreg:.3f}, AUPR: {aupr_logreg:
```

```
AdaBoost - AUROC: 0.674, AUPR: 0.786, AUPRG: 6.425
Logistic Regression - AUROC: 0.767, AUPR: 0.892, AUPRG: 13.128
```

The PR Gain curves provide a more meaningful assessment than standard PR curves in cases of imbalanced data, as they account for the base rate. In this case, LR outperformed AdaBoost across all metrics. The significantly higher AUPRG for LR (13.128) compared to AdaBoost (6.425) highlights that it is better suited for this task when considering the base rate. Therefore I agree with the NIPS paper's conclusion that practitioners should prefer PR Gain curves over PR curves for more accurate evaluation of model performance.

Problem 3

In [ ]:
```
import torch
import torchvision
import torchvision.transforms as transforms
import torch.nn as nn
import torch.optim as optim
import numpy as np
import matplotlib.pyplot as plt
from torch.optim.lr_scheduler import CyclicLR
%matplotlib inline
from torchsummary import summary

# Load FashionMNIST dataset
transform = transforms.Compose([transforms.ToTensor(), transforms.Normalize(

trainset = torchvision.datasets.FashionMNIST(root='./data', train=True, dowr
trainloader = torch.utils.data.DataLoader(trainset, batch_size=64, shuffle=T

testset = torchvision.datasets.FashionMNIST(root='./data', train=False, dowr
testloader = torch.utils.data.DataLoader(testset, batch_size=64, shuffle=Fal
```

```
Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/train
-images-idx3-ubyte.gz
Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/train
-images-idx3-ubyte.gz to ./data/FashionMNIST/raw/train-images-idx3-ubyte.gz
100%|██████████| 26421880/26421880 [00:03<00:00, 8502762.17it/s]
Extracting ./data/FashionMNIST/raw/train-images-idx3-ubyte.gz to ./data/Fash
ionMNIST/raw

Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/train
-labels-idx1-ubyte.gz
Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/train
-labels-idx1-ubyte.gz to ./data/FashionMNIST/raw/train-labels-idx1-ubyte.gz
100%|██████████| 29515/29515 [00:00<00:00, 208732.53it/s]
Extracting ./data/FashionMNIST/raw/train-labels-idx1-ubyte.gz to ./data/Fash
ionMNIST/raw

Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/t10k-
images-idx3-ubyte.gz
Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/t10k-
images-idx3-ubyte.gz to ./data/FashionMNIST/raw/t10k-images-idx3-ubyte.gz
100%|██████████| 4422102/4422102 [00:01<00:00, 2599474.81it/s]
Extracting ./data/FashionMNIST/raw/t10k-images-idx3-ubyte.gz to ./data/Fashi
onMNIST/raw

Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/t10k-
labels-idx1-ubyte.gz
Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/t10k-
labels-idx1-ubyte.gz to ./data/FashionMNIST/raw/t10k-labels-idx1-ubyte.gz
100%|██████████| 5148/5148 [00:00<00:00, 7786612.69it/s]
```

```
Extracting ./data/FashionMNIST/raw/t10k-labels-idx1-ubyte.gz to ./data/Fashi
onMNIST/raw
```

In [ ]:
```
class ConvModule(nn.Module):
    def __init__(self, in_channels, out_channels, kernel_size, stride, paddi
        super(ConvModule, self).__init__()
        self.conv = nn.Conv2d(in_channels, out_channels, kernel_size, stride
        self.bn = nn.BatchNorm2d(out_channels)
        self.act = nn.ReLU()

    def forward(self, x):
        x = self.conv(x)
        x = self.bn(x)
        x = self.act(x)
        return x

class InceptionModule(nn.Module):
    def __init__(self, in_channels, f_1x1, f_3x3):
        super(InceptionModule, self).__init__()
        self.branch1 = nn.Sequential(ConvModule(in_channels, f_1x1, kernel_s
        self.branch2 = nn.Sequential(ConvModule(in_channels, f_3x3, kernel_s

    def forward(self, x):
        branch1 = self.branch1(x)
        branch2 = self.branch2(x)
        return torch.cat([branch1, branch2], 1)

class DownsampleModule(nn.Module):
    def __init__(self, in_channels, f_3x3):
        super(DownsampleModule, self).__init__()
        self.branch1 = nn.Sequential(ConvModule(in_channels, f_3x3, kernel_s
        self.branch2 = nn.MaxPool2d(3, stride=2)

    def forward(self, x):
        branch1 = self.branch1(x)
        branch2 = self.branch2(x)
        return torch.cat([branch1, branch2], 1)

class InceptionSmall(nn.Module):
    def __init__(self, num_classes=10):
        super(InceptionSmall, self).__init__()
        self.conv1 = ConvModule(1, 96, 3, 1, 0)  # FashionMNIST has 1 input
        self.inception1 = InceptionModule(96, 32, 32)
        self.inception2 = InceptionModule(64, 32, 48)
        self.down1 = DownsampleModule(80, 80)
        self.inception3 = InceptionModule(160, 112, 48)
        self.inception4 = InceptionModule(160, 96, 64)
        self.inception5 = InceptionModule(160, 80, 80)
        self.inception6 = InceptionModule(160, 48, 96)
        self.down2 = DownsampleModule(144, 96)
        self.inception7 = InceptionModule(240, 176, 160)
        self.inception8 = InceptionModule(336, 176, 160)
        self.meanpool = nn.AdaptiveAvgPool2d((7, 7))
        self.fc = nn.Linear(16464, num_classes)
```

```
    def forward(self, x):
        x = self.conv1(x)
        x = self.inception1(x)
        x = self.inception2(x)
        x = self.down1(x)
        x = self.inception3(x)
        x = self.inception4(x)
        x = self.inception5(x)
        x = self.inception6(x)
        x = self.down2(x)
        x = self.inception7(x)
        x = self.inception8(x)
        x = self.meanpool(x)
        x = torch.flatten(x, 1)
        x = self.fc(x)
        return x
```

In [ ]:

```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# Initialize the model, loss function, and optimizer
net = InceptionSmall().to(device)
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(net.parameters(), lr=0.01, momentum=0.9)

# lr cycles between 1e-9 and 10
scheduler = torch.optim.lr_scheduler.CyclicLR(optimizer, base_lr=1e-9, max_l

# 5 epochs
epochs = 5
losses = []
lrs = []

for epoch in range(epochs):
    running_loss = 0.0
    for i, data in enumerate(trainloader, 0):
        inputs, labels = data[0].to(device), data[1].to(device)
        optimizer.zero_grad()

        #Forward pass
        outputs = net(inputs)
        loss = criterion(outputs, labels)

        #Backward pass and optim
        loss.backward()
        optimizer.step()
        scheduler.step()  #Update lr after batch

        #Record the loss and learning rate
        running_loss += loss.item()
        losses.append(running_loss / (i+1))
        lrs.append(optimizer.param_groups[0]['lr'])

        #print after 100 minibatches
        if i % 100 == 99:
            print(f"[Epoch {epoch+1}, Batch {i+1}] Loss: {running_loss / 100
            running_loss = 0.0
```

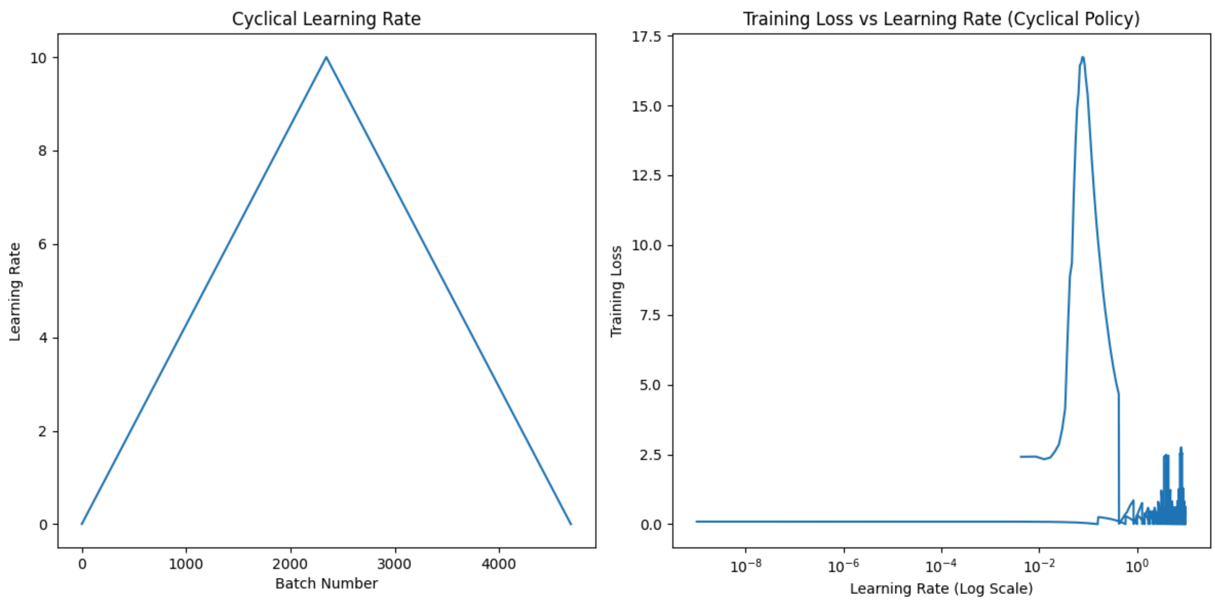
```
print("Training Complete")
```

```
[Epoch 1, Batch 100] Loss: 4.6578, LR: 0.4264392334
[Epoch 1, Batch 200] Loss: 1.7083, LR: 0.8528784657
[Epoch 1, Batch 300] Loss: 2.2855, LR: 1.2793176981
[Epoch 1, Batch 400] Loss: 2.3551, LR: 1.7057569305
[Epoch 1, Batch 500] Loss: 2.3745, LR: 2.1321961628
[Epoch 1, Batch 600] Loss: 2.3747, LR: 2.5586353952
[Epoch 1, Batch 700] Loss: 2.3867, LR: 2.9850746276
[Epoch 1, Batch 800] Loss: 2.4012, LR: 3.4115138599
[Epoch 1, Batch 900] Loss: 2.4069, LR: 3.8379530923
[Epoch 2, Batch 100] Loss: 2.4551, LR: 4.4264392330
[Epoch 2, Batch 200] Loss: 2.4524, LR: 4.8528784653
[Epoch 2, Batch 300] Loss: 2.4522, LR: 5.2793176977
[Epoch 2, Batch 400] Loss: 2.4505, LR: 5.7057569301
[Epoch 2, Batch 500] Loss: 2.4403, LR: 6.1321961624
[Epoch 2, Batch 600] Loss: 2.4569, LR: 6.5586353948
[Epoch 2, Batch 700] Loss: 2.5243, LR: 6.9850746272
[Epoch 2, Batch 800] Loss: 2.5225, LR: 7.4115138595
[Epoch 2, Batch 900] Loss: 2.5537, LR: 7.8379530919
[Epoch 3, Batch 100] Loss: 2.5374, LR: 8.4264392326
[Epoch 3, Batch 200] Loss: 2.5631, LR: 8.8528784649
[Epoch 3, Batch 300] Loss: 2.4872, LR: 9.2793176973
[Epoch 3, Batch 400] Loss: 2.5688, LR: 9.7057569297
[Epoch 3, Batch 500] Loss: 2.5643, LR: 9.8678038380
[Epoch 3, Batch 600] Loss: 2.6350, LR: 9.4413646056
[Epoch 3, Batch 700] Loss: 2.5255, LR: 9.0149253732
[Epoch 3, Batch 800] Loss: 2.5457, LR: 8.5884861409
[Epoch 3, Batch 900] Loss: 2.4993, LR: 8.1620469085
[Epoch 4, Batch 100] Loss: 2.5234, LR: 7.5735607678
[Epoch 4, Batch 200] Loss: 2.5016, LR: 7.1471215355
[Epoch 4, Batch 300] Loss: 2.4936, LR: 6.7206823031
[Epoch 4, Batch 400] Loss: 2.4563, LR: 6.2942430707
[Epoch 4, Batch 500] Loss: 2.4607, LR: 5.8678038384
[Epoch 4, Batch 600] Loss: 2.4564, LR: 5.4413646060
[Epoch 4, Batch 700] Loss: 2.4566, LR: 5.0149253736
[Epoch 4, Batch 800] Loss: 2.4877, LR: 4.5884861413
[Epoch 4, Batch 900] Loss: 2.4664, LR: 4.1620469089
[Epoch 5, Batch 100] Loss: 2.4161, LR: 3.5735607682
[Epoch 5, Batch 200] Loss: 2.4155, LR: 3.1471215359
[Epoch 5, Batch 300] Loss: 2.4156, LR: 2.7206823035
[Epoch 5, Batch 400] Loss: 2.3743, LR: 2.2942430711
[Epoch 5, Batch 500] Loss: 2.3732, LR: 1.8678038388
[Epoch 5, Batch 600] Loss: 2.3536, LR: 1.4413646064
[Epoch 5, Batch 700] Loss: 2.3413, LR: 1.0149253740
[Epoch 5, Batch 800] Loss: 2.3252, LR: 0.5884861417
[Epoch 5, Batch 900] Loss: 2.3210, LR: 0.1620469093
Training Complete
```

In [ ]:

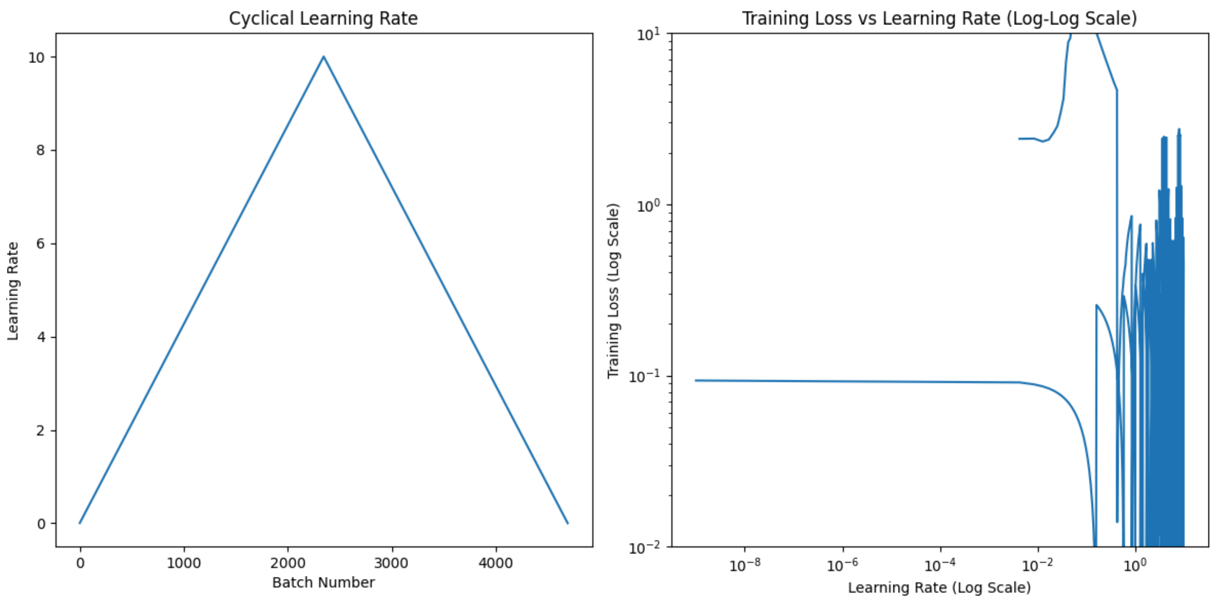
```
import matplotlib.pyplot as plt
import numpy as np

#Plot loss as function of LR
plt.figure(figsize=(12, 6))
```

```
plt.subplot(1, 2, 1)
plt.plot(lrs)
plt.xlabel("Batch Number")
plt.ylabel("Learning Rate")
plt.title("Cyclical Learning Rate")


plt.subplot(1, 2, 2)
plt.plot(lrs, losses)
plt.xscale('log')
plt.xlabel("Learning Rate (Log Scale)")
plt.ylabel("Training Loss")
plt.title("Training Loss vs Learning Rate (Cyclical Policy)")

plt.tight_layout()
plt.show()
```



In [ ]:
```
import matplotlib.pyplot as plt
import numpy as np

filtered_indices = [i for i, lr in enumerate(lrs) if 1e-9 <= lr <= 10]
filtered_lrs = np.array(lrs)[filtered_indices]
filtered_losses = np.array(losses)[filtered_indices]
plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1)
plt.plot(lrs)
plt.xlabel("Batch Number")
plt.ylabel("Learning Rate")
plt.title("Cyclical Learning Rate")

plt.subplot(1, 2, 2)
plt.plot(filtered_lrs, filtered_losses)
plt.xscale('log')
plt.yscale('log')
plt.xlabel("Learning Rate (Log Scale)")
```

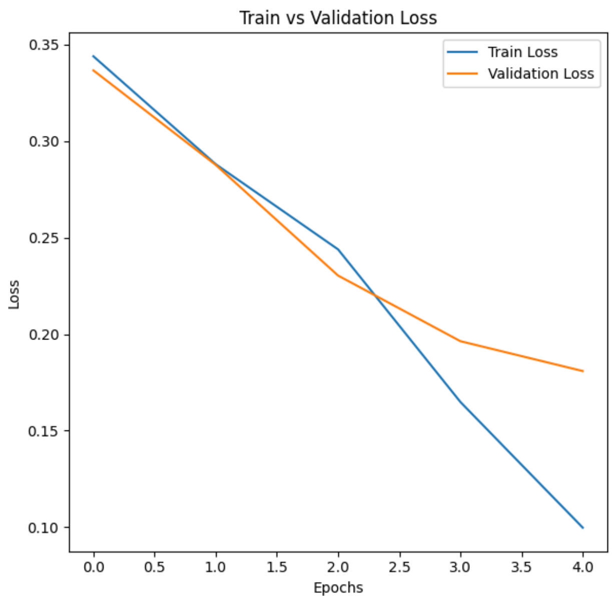
```
plt.ylabel("Training Loss (Log Scale)")
plt.title("Training Loss vs Learning Rate (Log-Log Scale)")
plt.ylim(0.01, 10)
plt.tight_layout()
plt.show()
```

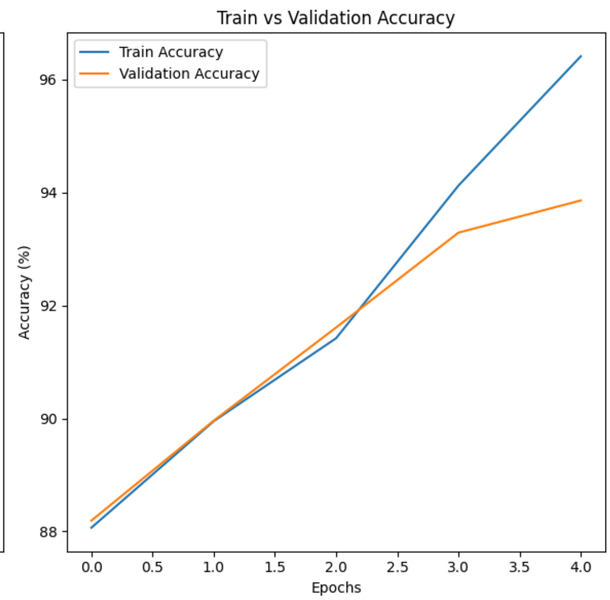


In [ ]:
```
# cyclical learning rate between 1e-4 and 1e-2

# Initialize optim/ scheduler
optimizer = torch.optim.SGD(net.parameters(), lr=0.01, momentum=0.9)
scheduler = torch.optim.lr_scheduler.CyclicLR(optimizer, base_lr=1e-4, max_l

# train for 5 epochs
epochs = 5
train_losses = []
val_losses = []
train_acc = []
val_acc = []

for epoch in range(epochs):
    running_train_loss = 0.0
    correct_train = 0
    total_train = 0
    net.train()

    #Train
    for i, data in enumerate(trainloader, 0):
        inputs, labels = data[0].to(device), data[1].to(device)
        optimizer.zero_grad()

        #Forward pass
        outputs = net(inputs)
        loss = criterion(outputs, labels)

        #Backward pass
        loss.backward()
        optimizer.step()
```

```
        scheduler.step()  #Update lr after batch

        running_train_loss += loss.item()
        _, predicted = outputs.max(1)
        total_train += labels.size(0)
        correct_train += predicted.eq(labels).sum().item()

    #Train acc loss
    train_losses.append(running_train_loss / len(trainloader))
    train_acc.append(100. * correct_train / total_train)

    #Val loop
    net.eval()
    correct_val = 0
    total_val = 0
    val_loss = 0.0
    with torch.no_grad():
        for data in testloader:
            inputs, labels = data[0].to(device), data[1].to(device)
            outputs = net(inputs)
            loss = criterion(outputs, labels)
            val_loss += loss.item()
            _, predicted = outputs.max(1)
            total_val += labels.size(0)
            correct_val += predicted.eq(labels).sum().item()

    val_losses.append(val_loss / len(testloader))
    val_acc.append(100. * correct_val / total_val)
    print(f"Epoch {epoch+1}/{epochs} | Train Loss: {train_losses[-1]:.4f}, T

# Plotting results
plt.figure(figsize=(12, 6))

# Plot train vs val loss
plt.subplot(1, 2, 1)
plt.plot(train_losses, label="Train Loss")
plt.plot(val_losses, label="Validation Loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.title("Train vs Validation Loss")
plt.legend()

#Plot train vs val acc
plt.subplot(1, 2, 2)
plt.plot(train_acc, label="Train Accuracy")
plt.plot(val_acc, label="Validation Accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy (%)")
plt.title("Train vs Validation Accuracy")
plt.legend()

plt.tight_layout()
plt.show()
```

```
Epoch 1/5 | Train Loss: 0.3439, Train Acc: 88.06% | Val Loss: 0.3366, Val Ac
c: 88.19%
Epoch 2/5 | Train Loss: 0.2880, Train Acc: 89.95% | Val Loss: 0.2877, Val Ac
c: 89.96%
Epoch 3/5 | Train Loss: 0.2439, Train Acc: 91.42% | Val Loss: 0.2303, Val Ac
c: 91.61%
Epoch 4/5 | Train Loss: 0.1648, Train Acc: 94.12% | Val Loss: 0.1963, Val Ac
c: 93.29%
Epoch 5/5 | Train Loss: 0.0997, Train Acc: 96.41% | Val Loss: 0.1809, Val Ac
c: 93.86%
```

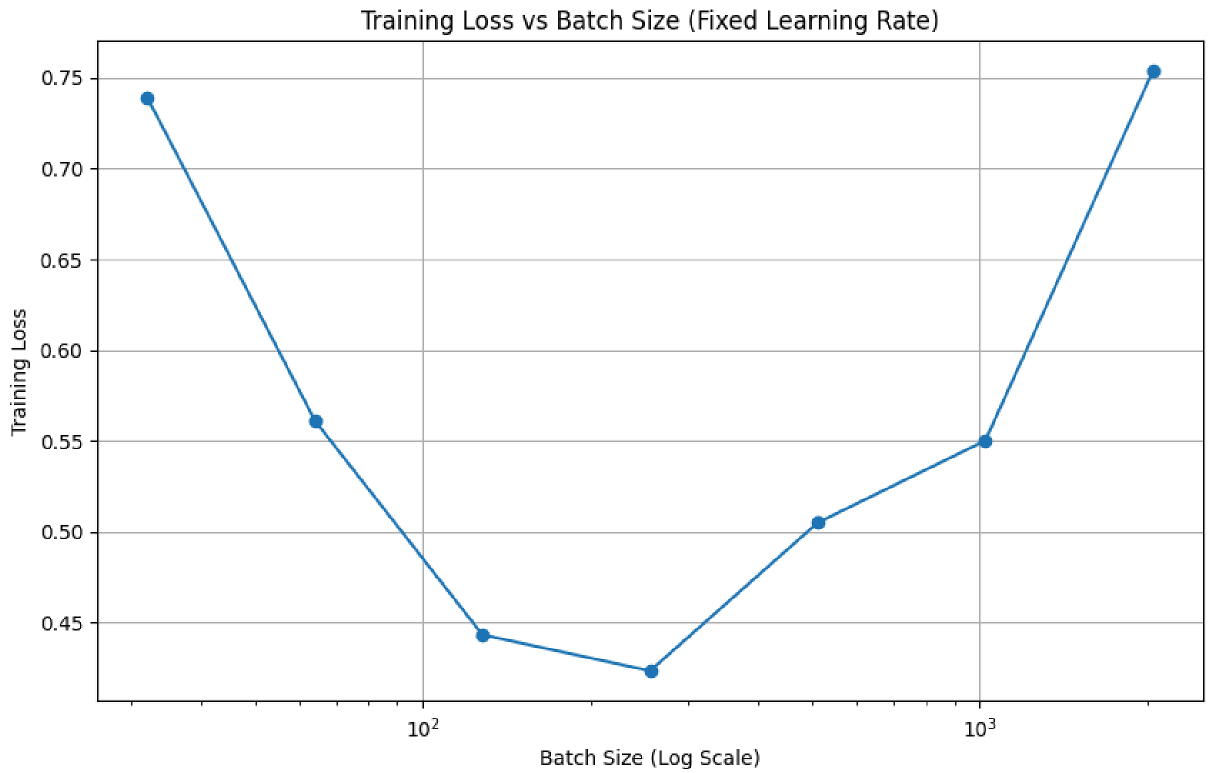




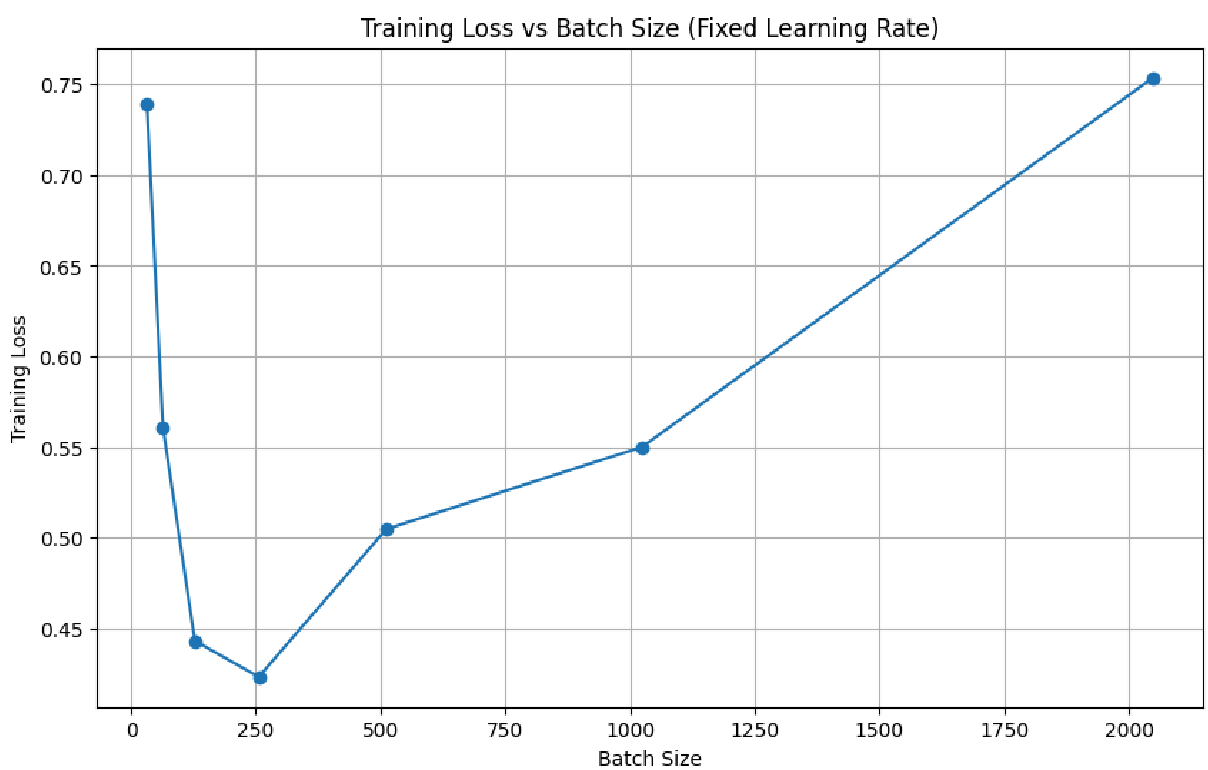
In [ ]:
```
import torch
import torch.optim as optim
import matplotlib.pyplot as plt

# Function to train with varying batch sizes
def train_with_increasing_batch_sizes(lrmax, initial_batch_size, max_batch_s
    batch_sizes = []
    train_losses = []

    for epoch in range(epoch_count):
        # double batch size
        current_batch_size = initial_batch_size * (2 ** epoch)
        if current_batch_size > max_batch_size:
            break

        # new dataloader
        trainloader = torch.utils.data.DataLoader(trainset, batch_size=curre

        # init model with fixed lr
        net = InceptionSmall().to(device)
        optimizer = optim.SGD(net.parameters(), lr=lrmax, momentum=0.9)
        criterion = nn.CrossEntropyLoss()

        running_loss = 0.0

        #train
        for i, data in enumerate(trainloader, 0):
```

```
            inputs, labels = data[0].to(device), data[1].to(device)
            optimizer.zero_grad()

            outputs = net(inputs)
            loss = criterion(outputs, labels)

            loss.backward()
            optimizer.step()

            running_loss += loss.item()

        #Calculate average loss
        avg_loss = running_loss / len(trainloader)
        train_losses.append(avg_loss)
        batch_sizes.append(current_batch_size)

        print(f"Epoch {epoch + 1}, Batch Size: {current_batch_size}, Train L

    return batch_sizes, train_losses

# Parameters
lrmax = 1e-2
initial_batch_size = 32
max_batch_size = 4096
epoch_count = 7

batch_sizes, train_losses = train_with_increasing_batch_sizes(lrmax, initial

plt.figure(figsize=(10, 6))
plt.plot(batch_sizes, train_losses, marker='o')
plt.xscale('log')
plt.xlabel('Batch Size (Log Scale)')
plt.ylabel('Training Loss')
plt.title('Training Loss vs Batch Size (Fixed Learning Rate)')
plt.grid(True)
plt.show()
```

```
Epoch 1, Batch Size: 32, Train Loss: 0.7390
Epoch 2, Batch Size: 64, Train Loss: 0.5611
Epoch 3, Batch Size: 128, Train Loss: 0.4432
Epoch 4, Batch Size: 256, Train Loss: 0.4234
Epoch 5, Batch Size: 512, Train Loss: 0.5049
Epoch 6, Batch Size: 1024, Train Loss: 0.5501
Epoch 7, Batch Size: 2048, Train Loss: 0.7536
```



In [ ]:
```
plt.figure(figsize=(10, 6))
plt.plot(batch_sizes, train_losses, marker='o')
plt.xlabel('Batch Size')
plt.ylabel('Training Loss')
plt.title('Training Loss vs Batch Size (Fixed Learning Rate)')
plt.grid(True)
plt.show()
```



The graph and results for the varying batch sizes show that training loss decreases as the batch size increases initially, reaching a minimum at batch size 256 with a loss of 0.4234. However, as the batch size continues to increase, the training loss begins to rise again, peaking at 0.7536 for a batch size of 2048. This suggests that while smaller batch sizes may initially lead to higher losses, very large batch sizes do not necessarily yield better performance and can, in fact, hurt generalization. The cyclical learning rate approach is a more balanced generalization, especially with moderate batch sizes like 64/128/256.

In [ ]:

VGG19 Memory and Weights

Layer	**Number of Activations (Memory)**	**Parameters (Compute)**
Input	$224 \times 224 \times 3 = 150\,K$	0
CONV3-64	$224 \times 224 \times 64 = 3.2\,M$	$(3 \times 3 \times 3) \times 64 = 1,728$
CONV3-64	$224 \times 224 \times 64 = 3.2\,M$	$(3 \times 3 \times 64) \times 64 = 36,864$
POOL2	$112 \times 112 \times 64 = 800\,K$	0
CONV3-128	$112 \times 112 \times 128 = 1.6\,M$	$(3 \times 3 \times 64) \times 128 = 73,728$
CONV3-128	$112 \times 112 \times 128 = 1.6\,M$	$(3 \times 3 \times 128) \times 128 = 147,456$
POOL2	$56 \times 56 \times 128 = 400\,K$	0
CONV3-256	$56 \times 56 \times 256 = 800\,K$	$(3 \times 3 \times 128) \times 256 = 294,912$
CONV3-256	$56 \times 56 \times 256 = 800\,K$	$(3 \times 3 \times 256) \times 256 = 589,824$
CONV3-256	$56 \times 56 \times 256 = 800\,K$	$(3 \times 3 \times 256) \times 256 = 589,824$
CONV3-256	$56 \times 56 \times 256 = 800\,K$	$(3 \times 3 \times 256) \times 256 = 589,824$
POOL2	$28 \times 28 \times 256 = 200\,K$	0
CONV3-512	$28 \times 28 \times 512 = 400\,K$	$(3 \times 3 \times 256) \times 512 = 1,179,648$
CONV3-512	$28 \times 28 \times 512 = 400\,K$	$(3 \times 3 \times 512) \times 512 = 2,359,296$
CONV3-512	$28 \times 28 \times 512 = 400\,K$	$(3 \times 3 \times 512) \times 512 = 2,359,296$
CONV3-512	$28 \times 28 \times 512 = 400\,K$	$(3 \times 3 \times 512) \times 512 = 2,359,296$
POOL2	$14 \times 14 \times 512 = 100\,K$	0
CONV3-512	$14 \times 14 \times 512 = 100\,K$	$(3 \times 3 \times 512) \times 512 = 2,359,296$
CONV3-512	$14 \times 14 \times 512 = 100\,K$	$(3 \times 3 \times 512) \times 512 = 2,359,296$
CONV3-512	$14 \times 14 \times 512 = 100\,K$	$(3 \times 3 \times 512) \times 512 = 2,359,296$
CONV3-512	$14 \times 14 \times 512 = 100\,K$	$(3 \times 3 \times 512) \times 512 = 2,359,296$
POOL2	$7 \times 7 \times 512 = 25\,K$	0
FC	4096	$4096 \times 4096 = 16,777,216$
FC	4096	$4096 \times 4096 = 16,777,216$
FC	1000	$4096 \times 1000 = 4,096,000$
Total Activations	$17.62\,M$	**Total Parameters** $= 57,668,812$

(a)

The inception module in cnns is designed to capture multi-scale information by using parallel convolutional filters of different sizes (1x1, 3x3, 5x5) along with pooling operations. These filters operate in parallel, allowing the network to learn both local and global features effectively. The outputs from these filters are concatenated, preserving the spatial dimensions while increasing the depth of the feature maps.

(b)

Naive Inception Module:

- **1x1 Convolutions**: $32 \times 32 \times 128$
- **3x3 Convolutions**: $32 \times 32 \times 192$

- **5x5 Convolutions**: $32 \times 32 \times 96$
- **3x3 Max Pooling**: $32 \times 32 \times 256$

$$\text{Total Output Size} = 32 \times 32 \times (128 + 192 + 96 + 256) = 32 \times 32 \times 672$$

Inception Module with Dimension Reduction:

- **1x1 Convolutions**: $32 \times 32 \times 128$
- **3x3 Convolutions** (with 1x1 reduction): $32 \times 32 \times 192$
- **5x5 Convolutions** (with 1x1 reduction): $32 \times 32 \times 96$
- **1x1 Convolution after Max Pooling**: $32 \times 32 \times 64$

$$\text{Total Output Size} = 32 \times 32 \times (128 + 192 + 96 + 64) = 32 \times 32 \times 480$$

(c)

Naive Inception Module:

- **1x1 Convolutions**:

$$1 \times 1 \times 256 \times 128 \times 32 \times 32 = 335,544,32$$

- **3x3 Convolutions**:

$$3 \times 3 \times 256 \times 192 \times 32 \times 32 = 1,131,524,096$$

- **5x5 Convolutions**:

$$5 \times 5 \times 256 \times 96 \times 32 \times 32 = 983,040,000$$

$$\text{Total Operations} = 2,450,136,128$$

Inception Module with Dimension Reduction:

- **1x1 Convolutions for Reduction**:

$$1 \times 1 \times 256 \times 128 \times 32 \times 32 = 335,544,32$$

- **3x3 Convolutions (after reduction)**:

$$3 \times 3 \times 128 \times 192 \times 32 \times 32 = 566,362,24$$

- **5x5 Convolutions (after reduction)**:

$$5 \times 5 \times 32 \times 96 \times 32 \times 32 = 122,880,00$$

$$\text{Total Operations} = 1,024,726,56$$

(d)

The dimension reduction version reduces the computational complexity by approximately 58.4%, making it significantly more efficient than the naive version.

$$\text{Computational Savings} = \frac{2.45 - 1.02}{2.45} \times 100 \approx 58.4\%$$

Problem 5 - Staleness

1. **Gradient (g[L1, 1])**:
   - No gradients from Learner 2.
   - **Staleness**: (0)
2. **Gradient (g[L1, 2])**:
   - No gradients from Learner 2.
   - **Staleness**: (0)
3. **Gradient (g[L1, 3])**:
   - Learner 2 has calculated (g[L2, 1]).
   - **Staleness**: (1)
4. **Gradient (g[L1, 4])**:
   - Learner 2 has computed (g[L2, 1]).
   - **Staleness**: (1)
5. **Gradient (g[L2, 1])**:
   - Learner 2 sends (g[L2, 1]) at second 2.5. Learner 1 has computed g[L1, 1] and g[L1, 2], which both updated the weights.
   - **Staleness**: (2)
6. **Gradient (g[L2, 2])**:
   - Learner 2 sends g[L2, 2] at second 5. Learner 1 has sent g[L1, 3] and g[L1, 4], which updated the weights.
   - **Staleness**: (2)

Answer:

- (g[L1, 1]): 0 updates
- (g[L1, 2]): 0 updates
- (g[L1, 3]): 1 update
- (g[L1, 4]): 1 update
- (g[L2, 1]): 2 updates
- (g[L2, 2]): 2 updates