CEG5306 Homework 3

Zhao Lixiuqi

A0239866J

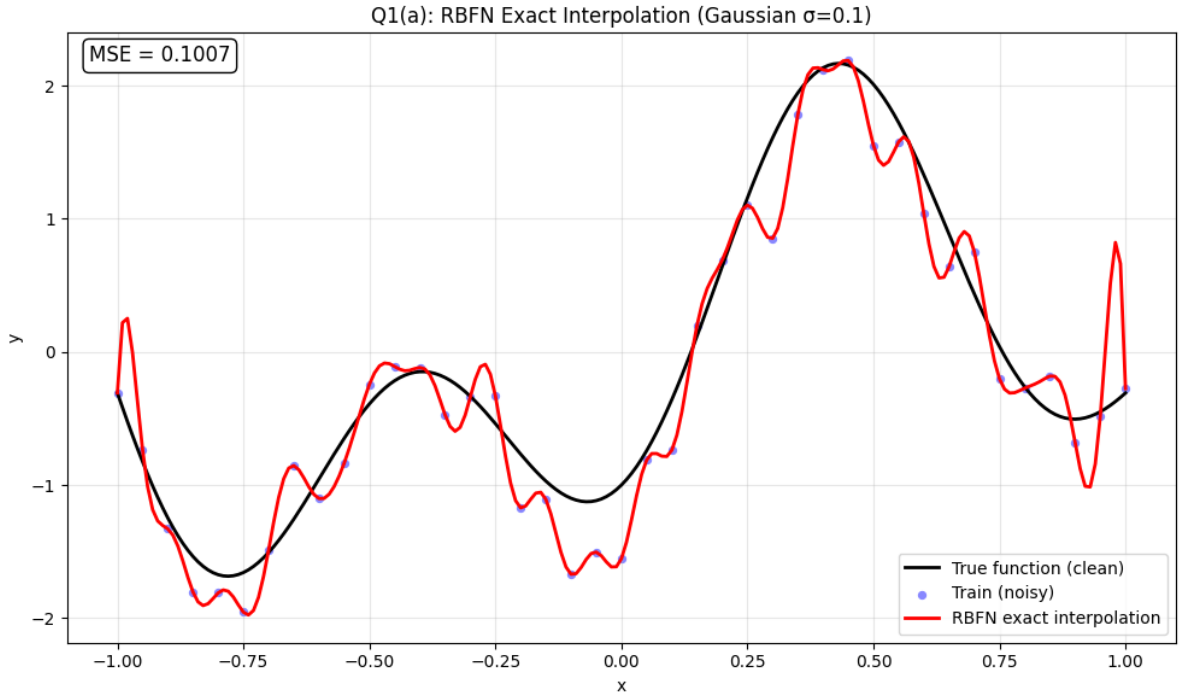
Qn a:
code for RBFN with exact interpolation:

```
import torch
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import mean_squared_error

torch.set_default_dtype(torch.float64)

def true_func_t(x: torch.Tensor) -> torch.Tensor:
    return 1.2 * torch.sin(torch.pi * x) - torch.cos(2.4 * torch.pi * x)

def make_data_t(
    x_min=-1.0,
    x_max=1.0,
    step_train=0.05,
    step_test=0.01,
    noise_std=0.3,
    seed=7,
):
    rng = np.random.default_rng(seed)
    x_train = torch.tensor(np.arange(x_min, x_max + 1e-12, step_train))
    y_clean_train = true_func_t(x_train)
    noise = torch.tensor(rng.normal(0.0, 1.0, size=x_train.shape))
    y_train = y_clean_train + noise_std * noise

    x_test = torch.tensor(np.arange(x_min, x_max + 1e-12, step_test))
    y_test = true_func_t(x_test)  # clean for fair evaluation
    return x_train, y_train, x_test, y_test

def rbf_gaussian_t(x: torch.Tensor, c: torch.Tensor, sigma: float) -> torch.Tensor:
    d2 = (x[:, None] - c[None, :]) ** 2
    return torch.exp(-d2 / (2.0 * sigma**2))

def solve_exact_interpolation_t(
    x_train: torch.Tensor,
    y_train: torch.Tensor,
    sigma: float,
    ridge: float = 0.0,
):
    centers = x_train.clone()
    Phi = rbf_gaussian_t(x_train, centers, sigma)
    if ridge > 0:
        G = Phi.T @ Phi + ridge * torch.eye(Phi.shape[1], dtype=Phi.dtype)
```

```
        b = Phi.T @ y_train
        w = torch.linalg.solve(G, b)
    else:
        w = torch.linalg.solve(Phi, y_train)
    return centers, w

def predict_t(x: torch.Tensor, centers: torch.Tensor, w: torch.Tensor, sigma:
float):
    Phi = rbf_gaussian_t(x, centers, sigma)
    return Phi @ w


def rmse(y_true_np: np.ndarray, y_pred_np: np.ndarray) -> float:
    return float(np.sqrt(mean_squared_error(y_true_np, y_pred_np)))

def main():
    x_tr, y_tr, x_te, y_te = make_data_t()
    sigma = 0.1  # given

    centers, w = solve_exact_interpolation_t(
        x_tr, y_tr, sigma=sigma, ridge=0.0)

    yhat_tr = predict_t(x_tr, centers, w, sigma)
    yhat_te = predict_t(x_te, centers, w, sigma)

    mse_tr = mean_squared_error(y_tr.numpy(), yhat_tr.detach().numpy())
    mse_te = mean_squared_error(y_te.numpy(), yhat_te.detach().numpy())

    print("Q1(a) PyTorch - Exact Interpolation (σ = 0.1)")
    print(f"  Train MSE (noisy): {mse_tr:.6f}")
    print(f"  Test  MSE (clean): {mse_te:.6f}")
    print(f"  #centers: {len(centers)}")

    xx = x_te.numpy()
    yy_true = y_te.numpy()

    plt.figure(figsize=(10, 6))
    plt.plot(xx, yy_true, "k-", lw=2, label="True function (clean)")
    plt.scatter(x_tr.numpy(), y_tr.numpy(), s=18, color="#8888ff",
                label="Train (noisy)")
    plt.plot(xx, yhat_te.detach().numpy(), "r-", lw=2,
             label="RBFN exact interpolation"
             )

    plt.text(0.02, 0.98, f'MSE = {mse_te:.4f}',
             transform=plt.gca().transAxes,
             fontsize=12,
             verticalalignment='top',
             bbox=dict(boxstyle='round,pad=0.3', facecolor='white',
edgecolor='black'))
```

```
    plt.title("Q1(a): RBFN Exact Interpolation (Gaussian σ=0.1)")
    plt.xlabel("x")
    plt.ylabel("y")
    plt.legend()
    plt.grid(True, alpha=0.3)
    plt.tight_layout()
    plt.show()

if __name__ == "__main__":
    main()
```

plot obtained (MSE = 0.1007) :



As we can see from the result, the RBFN that uses exact interpolation:
1. Passes through all the noisy points generated during the training;

2. Is not able to generalise well to fit the true function due to the noisy training set.

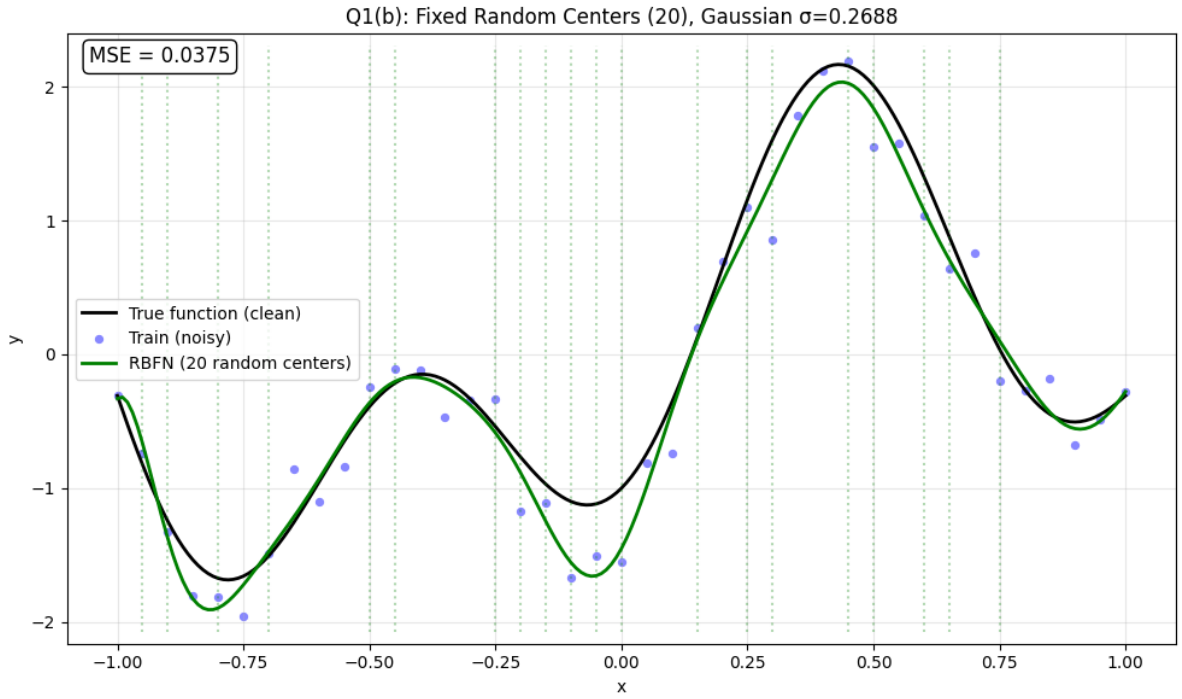
Qn b:
code for RBFN with the strategy of “Fixed Centers Selected at Random”:

```
import torch
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import mean_squared_error

torch.set_default_dtype(torch.float64)

def true_func_t(x: torch.Tensor) -> torch.Tensor:
    return 1.2 * torch.sin(torch.pi * x) - torch.cos(2.4 * torch.pi * x)

def make_data_t(
    x_min=-1.0,
    x_max=1.0,
    step_train=0.05,
    step_test=0.01,
    noise_std=0.3,
    seed=7,
):
    rng = np.random.default_rng(seed)
    x_train = torch.tensor(
        np.arange(x_min, x_max + 1e-12, step_train)
    )
    y_clean_train = true_func_t(x_train)
    noise = torch.tensor(rng.normal(0.0, 1.0, size=x_train.shape))
    y_train = y_clean_train + noise_std * noise

    x_test = torch.tensor(
        np.arange(x_min, x_max + 1e-12, step_test)
    )
    y_test = true_func_t(x_test)
    return x_train, y_train, x_test, y_test

def rbf_gaussian_t(
    x: torch.Tensor, c: torch.Tensor, sigma: float
) -> torch.Tensor:
    d2 = (x[:, None] - c[None, :]) ** 2
    return torch.exp(-d2 / (2.0 * sigma**2))


def solve_fixed_random_centers_t(
    x_train: torch.Tensor,
    y_train: torch.Tensor,
    num_centers: int = 20,
    ridge: float = 0.0,
    seed: int = 42,
):
    rng = np.random.default_rng(seed)
    idx = np.sort(rng.choice(len(x_train), num_centers, replace=False))
    centers = x_train[idx]
```

```
    # Calculate sigma using the formula: σ_i = d_max / sqrt(2M)
    # where d_max is the maximum distance between chosen centers
    centers_np = centers.numpy()
    distances = []
    for i in range(len(centers_np)):
        for j in range(i + 1, len(centers_np)):
            distances.append(abs(centers_np[i] - centers_np[j]))
    d_max = max(distances)
    sigma = d_max / np.sqrt(2 * num_centers)

    Phi = rbf_gaussian_t(x_train, centers, sigma)
    G = Phi.T @ Phi + ridge * torch.eye(Phi.shape[1], dtype=Phi.dtype)
    b = Phi.T @ y_train
    w = torch.linalg.solve(G, b)
    return centers, w, sigma

def predict_t(
    x: torch.Tensor, centers: torch.Tensor, w: torch.Tensor, sigma: float
) -> torch.Tensor:
    Phi = rbf_gaussian_t(x, centers, sigma)
    return Phi @ w

def main():
    x_tr, y_tr, x_te, y_te = make_data_t()
    num_centers = 20
    ridge = 0.0

    centers, w, sigma = solve_fixed_random_centers_t(
        x_train=x_tr,
        y_train=y_tr,
        num_centers=num_centers,
        ridge=ridge,
        seed=42,
    )

    yhat_tr = predict_t(x_tr, centers, w, sigma)
    yhat_te = predict_t(x_te, centers, w, sigma)

    mse_tr = mean_squared_error(y_tr.numpy(), yhat_tr.detach().numpy())
    mse_te = mean_squared_error(y_te.numpy(), yhat_te.detach().numpy())

    print("Q1(b) Fixed Random Centers — PyTorch")
    print(f"  #centers: {num_centers}")
    print(f"  Train MSE (noisy): {mse_tr:.6f}")
    print(f"  Test  MSE (clean): {mse_te:.6f}")

    xx = x_te.numpy()
    yy_true = y_te.numpy()
    yhat_te_np = yhat_te.detach().numpy()

    plt.figure(figsize=(10, 6))
```

```
    plt.plot(xx, yy_true, "k-", lw=2, label="True function (clean)")
    plt.scatter(
        x_tr.numpy(), y_tr.numpy(), s=18, color="#8888ff",
        label="Train (noisy)"
    )
    plt.plot(
        xx, yhat_te_np, "g-", lw=2,
        label=f"RBFN (20 random centers)"
    )

    for c in centers.numpy():
        plt.axvline(
            c, ymin=0.02, ymax=0.98,
            color="g", linestyle="dotted", alpha=0.3
        )

    plt.text(
        0.02, 0.98,
        f"MSE = {mse_te:.4f}",
        transform=plt.gca().transAxes,
        fontsize=12,
        verticalalignment="top",
        bbox=dict(
            boxstyle="round,pad=0.3",
            facecolor="white",
            edgecolor="black"
        )
    )

    plt.title(f"Q1(b): Fixed Random Centers (20), Gaussian σ={sigma:.4f}")
    plt.xlabel("x")
    plt.ylabel("y")
    plt.legend()
    plt.grid(True, alpha=0.3)
    plt.tight_layout()
    plt.show()

if __name__ == "__main__":
    main()
```

plot obtained (MSE = 0.0375):



The MSE (mean squared error) of the fixed random center approach is much smaller than that of the exact interpolation. (0.0375 < 0.1007)

As we can observe intuitively from the graph, the fixed random center approach generates a result graph that is much more smooth (less oscillations) graph as compared to that of the exact interpolation. This is due to the decrease in the number of neurons in the hidden layer and also a different approach of training. This shows that this approach is better in generalisation such that the characteristic of the true function is better approximated, instead of getting over-fitted result due to noisy training set.

Declaration of use of AI:
I used GPT5 to help me with the use of pytorch and matplotlib library