Homework #3

CSE 446/546: Machine Learning

Ray Chen

Due: **Wednesday** Nov 23, 2022 11:59pm

A: 101 points, **B**: 16 points

A1:

- **Part a:**
  We should decrease σ. Ad the σ^2 is large, the shape would be very flat, which means the fit graph is very far from the data points. As we want to make the fit more accurate, we should decrease σ.

- **Part b:**
  True. Even gradient descent may not reach the globally-optimal solution, but minimizing a non-convex loss function can get the globally-optimal solution.

- **Part c:**
  False. Initialing all the weight to zero is not a good idea. If we initial all weights to zero, the neural network may repeat working on the saddle points.

- **Part d:**
  True. Non-linear activation function can make network learns non-linear decision boundaries. This allows the model to learn more complex functions than a network trained using a linear activation function.

- **Part e:**
  False. The time complexity of the backward pass step is the same as that of the forward pass step in a neural network.

- **Part f:**
  True. Because Neural networks can help computers make intelligent decisions with limited human assistance, which means they can learn the relationship between input and output even that is complex or nonlinear.

A2:

- **Part a:**

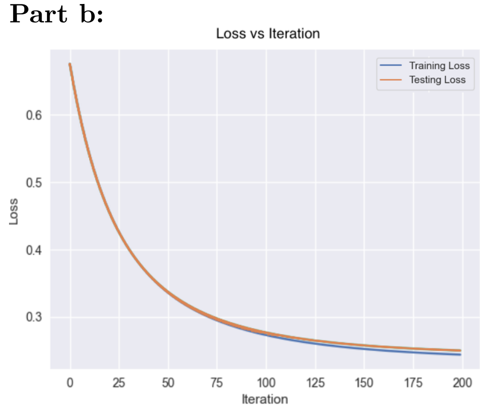
$$
\begin{aligned}
\nabla_w J(w,b) &= \nabla_w\Big(\frac{1}{n}\sum_{i=1}^{n}\log(1+\exp(-y_i(b+x_i^Tw))) + \lambda||w||_2^2\Big) \\
&= \frac{1}{n}\sum_{i=1}^{n}\nabla_w\log(1+\exp(-y_i(b+x_i^Tw))) + \nabla_w\lambda||w||_2^2 \\
&= \frac{1}{n}\sum_{i=1}^{n}\nabla_w - \log(\mu_i(w,b)) + \nabla_w\lambda||w||_2^2 \\
&= -\frac{1}{n}\sum_{i=1}^{n}\frac{\nabla_w\mu_i(w,b)}{\mu_i(w,b)} + 2\lambda w \\
&= -\frac{1}{n}\sum_{i=1}^{n}\nabla_w\frac{1}{1+\exp(-y_i(b+x_i^Tw))}\frac{1}{\mu_i(w,b)} + 2\lambda w \\
&= -\frac{1}{n}\sum_{i=1}^{n}\nabla_w\frac{-y_ix_i\exp(-y_i(b+x_i^Tw))}{(1+\exp(-y_i(b+x_i^Tw)))^2}\frac{1}{\mu_i(w,b)} + 2\lambda w \\
&= -\frac{1}{n}\sum_{i=1}^{n}\frac{-y_ix_i\frac{1-\mu_i(w,b)}{\mu_i(w,b)}}{\mu_i(w,b)^2}\frac{1}{\mu_i(w,b)} + 2\lambda w \\
&= -\frac{1}{n}\sum_{i=1}^{n}(-y_ix_i(1-\mu_i(w,b))) + 2\lambda w
\end{aligned}
$$

where we let $\mu_i(w,b) = \frac{1}{1+\exp(-y_i(b+x_i^Tw))}$

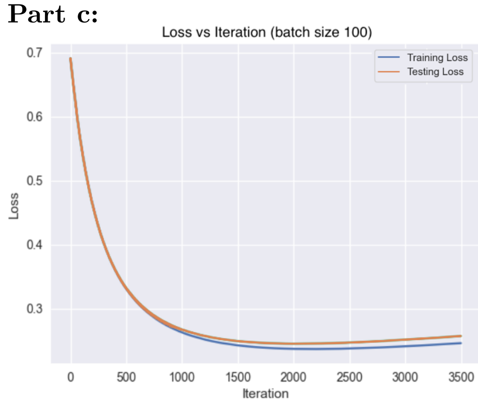
$$
\begin{aligned}
\nabla_b J(w,b) &= \nabla_w\Big(\frac{1}{n}\sum_{i=1}^{n}\log(1+\exp(-y_i(b+x_i^Tw))) + \lambda||w||_2^2\Big) \\
&= \frac{1}{n}\sum_{i=1}^{n}\nabla_b\log(1+\exp(-y_i(b+x_i^Tw))) \\
&= \frac{1}{n}\sum_{i=1}^{n}\nabla_b - \log(\mu_i(w,b)) \\
&= -\frac{1}{n}\sum_{i=1}^{n}\frac{\nabla_b\mu_i(w,b)}{\mu_i(w,b)} \\
&= -\frac{1}{n}\sum_{i=1}^{n}\nabla_b\frac{1}{1+\exp(-y_i(b+x_i^Tw))}\frac{1}{\mu_i(w,b)} \\
&= -\frac{1}{n}\sum_{i=1}^{n}\nabla_b\frac{-y_i\exp(-y_i(b+x_i^Tw))}{(1+\exp(-y_i(b+x_i^Tw)))^2}\frac{1}{\mu_i(w,b)} \\
&= -\frac{1}{n}\sum_{i=1}^{n}\frac{-y_i\frac{1-\mu_i(w,b)}{\mu_i(w,b)}}{\mu_i(w,b)^2}\frac{1}{\mu_i(w,b)} \\
&= -\frac{1}{n}\sum_{i=1}^{n}(-y_i(1-\mu_i(w,b)))
\end{aligned}
$$

where we let $\mu_i(w,b) = \frac{1}{1+\exp(-y_i(b+x_i^Tw))}$

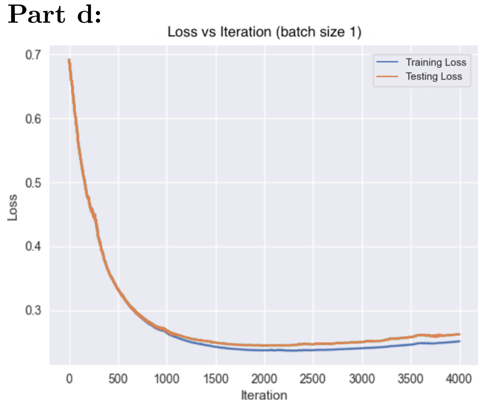
- **Part b:**

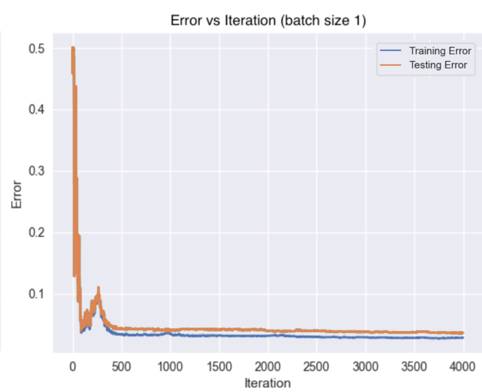


- **Part c:**



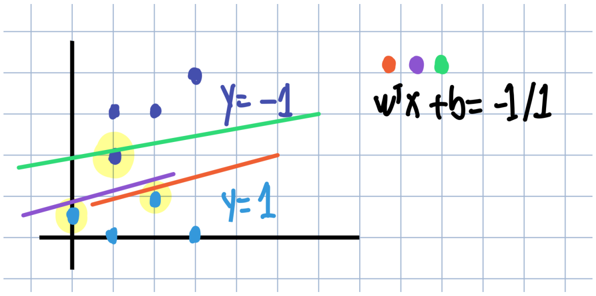
- **Part d:**





A3:

- **Part a:**



- **Part a:**

$$\begin{cases} w_1^T x + b &= -1 \\ w_2^T x + b &= 1 \\ w_2^T x + b &= 1 \end{cases} \Rightarrow \begin{cases} w_1^T \times 1 + b &= -1 \\ w_2^T \times 2 + b &= 1 \\ w_2^T \times 0 + b &= 1 \end{cases} \Rightarrow \begin{cases} w_1 &= -2 \\ w_2 &= 0 \\ b &= 1 \end{cases}$$

A4:

- **Part a:**

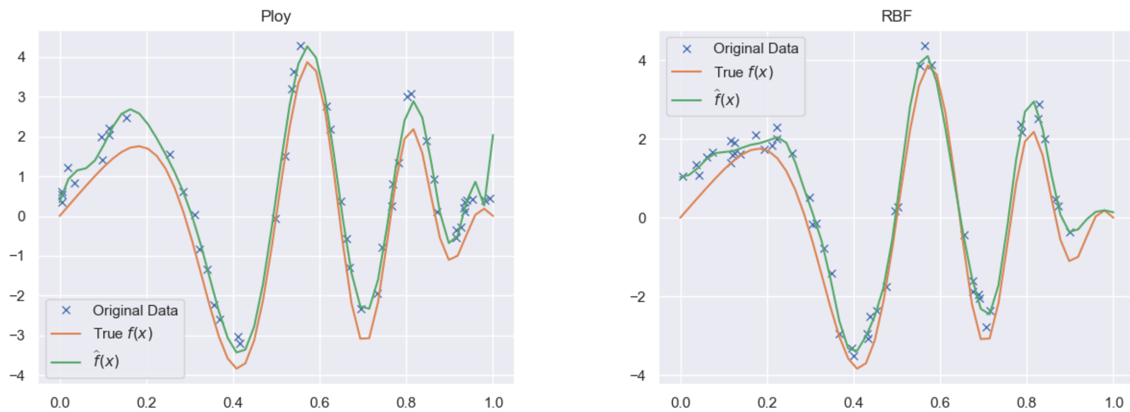
$$
\begin{aligned}
\phi(x) \cdot \phi(x') &= \sum_{i=0}^{\infty} (\frac{1}{\sqrt{i!}} e^{-\frac{x^2}{2}} x^i)(\frac{1}{\sqrt{i!}} e^{-\frac{x'^2}{2}} x'^i) \\
&= e^{-\frac{x^2 - x'^2}{2}} \sum_{i=0}^{\infty} \frac{1}{i!} x^i x'^i \\
&= e^{-\frac{x^2 - x'^2}{2}} e^{xx'} \\
&= e^{-\frac{x^2 + 2xx' - x'^2}{2}} \\
&= e^{-\frac{(x-x')^2}{2}}
\end{aligned}
$$

where we using Taylor expansion: $e^z = \sum_{n=0} \infty \frac{z^n}{n!}$

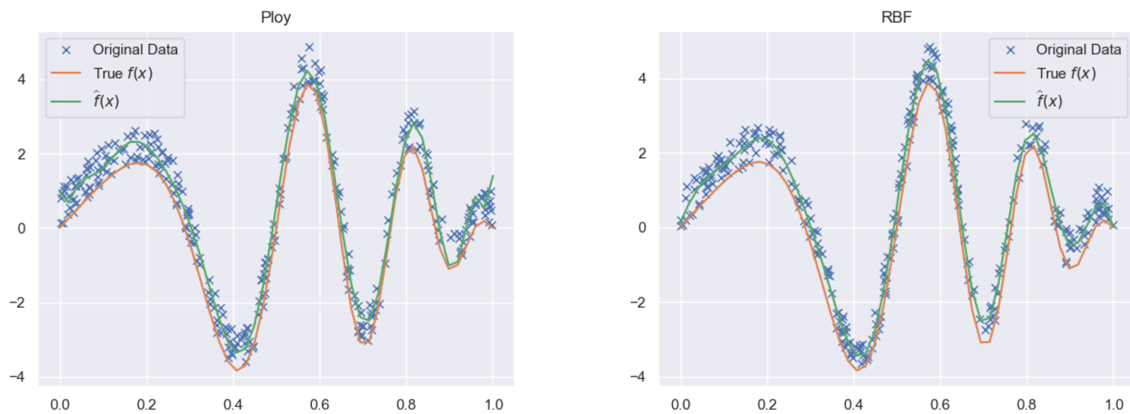
A5:

- **Part a:**
  Polynomial Kernel: $d = 23$ and $\lambda = 0.01$
  RBF Kernel: $\gamma = 3.5$ and $\lambda = 0.00001$

- **Part b:**



- **Part c:**
  Polynomial Kernel: $d = 14$ and $\lambda = 0.001$
  RBF Kernel: $\gamma = 2$ and $\lambda = 0.000001$



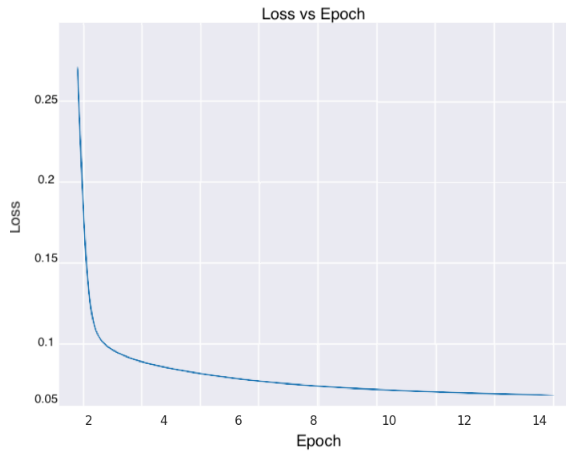
```
from typing import Tuple, Union
from scipy.linalg import solve
import matplotlib.pyplot as plt
import numpy as np

from utils import load_dataset, problem

def f_true(x: np.ndarray) -> np.ndarray:
    return 4 * np.sin(np.pi * x) * np.cos(6 * np.pi * x ** 2)

@problem.tag("hw3-A")
def poly_kernel(x_i: np.ndarray, x_j: np.ndarray, d: int) -> np.ndarray:
    return (1 + np.outer(x_i, x_j)) ** d

@problem.tag("hw3-A")
def rbf_kernel(x_i: np.ndarray, x_j: np.ndarray, gamma: float) -> np.ndarray:
    return np.exp(-gamma * np.subtract.outer(x_i, x_j) ** 2)

@problem.tag("hw3-A")
def train(
    x: np.ndarray,
    y: np.ndarray,
    kernel_function: Union[poly_kernel, rbf_kernel],  # type: ignore
    kernel_param: Union[int, float],
    _lambda: float,
) -> np.ndarray:
    gamma = 1. / np.median(np.subtract.outer(_x, _x) ** 2)
    if kernel_param is None:
        kernel_param = {
            'd': np.arange(7, 15),
            'lambda': [10 ** d for d in np.arange(-6, -1, 0.5)],
            'gamma': np.random.uniform(-5, 5, size=15) + gamma,
        }
        kernel_function = poly_kernel if kernel_function == 'poly' else rbf_kernel
    best_params = None
    params_hist = []
    for i in range(len(x)):
        if kernel_function == 'poly':
            poly_param_search(x,y,i)
        elif kernel_function == 'rbf':
            rbf_param_search(x,y,i)
    k_f = kernel_function(x_tr_i, _x[np.logical_not(idxs)], hyper[0])
    y_pred_i = np.matmul(alpha, k_f)
    loss += np.sum((y_pred_i - y[np.logical_not(idxs)]) ** 2)
    loss /= len(k)
    if best_params is None or best_params_loss > loss:
        best_params = hyper
        best_params_loss = loss
    params_hist.append(hyper)
    final_alpha = solve(kernel_function(_x, _x, best_params[0]) + best_params[1] * np.identity(len(_x)),y
    return final_alpha, best_params

@problem.tag("hw3-A", start_line=1)
def cross_validation(
    x: np.ndarray,
```

```
    y: np.ndarray,
    kernel_function: Union[poly_kernel, rbf_kernel],  # type: ignore
    kernel_param: Union[int, float],
    _lambda: float,
    num_folds: int,
) -> float:
    fold_size = len(x) // num_folds
    err = []
    for train_index, test_index in fold_size.split(x):
        X_train, X_test = x[train_index], x[test_index]
        Y_train, Y_test = y[train_index], y[test_index]
        kernel_function.fit(X_train, Y_train)
        Y_p = kernel_function.predict(X_test)
        err.append(np.mean((Y_p - Y_test) ** 2))
    return np.mean(err)

@problem.tag("hw3-A")
def rbf_param_search(
    x: np.ndarray, y: np.ndarray, num_folds: int
) -> Tuple[float, float]:
    for i in range(num_folds):
        if i == len(hyper_params['lambda']): break
        hyper = (np.random.choice(hyper_params['gamma']),np.random.choice(hyper_params['lambda']))
        while hyper in params_hist:
            hyper = (np.random.choice(hyper_params['gamma']),np.random.choice(hyper_params['lambda']))

@problem.tag("hw3-A")
def poly_param_search(
    x: np.ndarray, y: np.ndarray, num_folds: int
) -> Tuple[float, int]:
    for i in range(num_folds):
        if i == len(hyper_params['lambda']) * len(hyper_params['d']): break
        hyper =(np.random.choice(hyper_params['d']),np.random.choice(hyper_params['lambda']))
        while hyper in params_hist:
            hyper = (np.random.choice(hyper_params['d']),np.random.choice(hyper_params['lambda']))

@problem.tag("hw3-A", start_line=1)
def bootstrap(
    x: np.ndarray,
    y: np.ndarray,
    kernel_function: Union[poly_kernel, rbf_kernel],  # type: ignore
    kernel_param: Union[int, float],
    _lambda: float,
    bootstrap_iters: int = 300,
) -> np.ndarray:
    x_fine_grid = np.linspace(0, 1, 100)
    preds = []
    for i in range(bootstrap_iters):
        boot_index = np.random.choice(indices, size=x_fine_grid.shape[0], replace=True)
        X_boot, Y_boot = x_fine_grid[boot_index], y[boot_index]
        kernel_function.fit(X_boot, Y_boot)
        preds.append(kernel_function.predict(x))
    return preds

@problem.tag("hw3-A", start_line=1)
```

```python
def main():
    (x_30, y_30), (x_300, y_300), (x_1000, y_1000) = load_dataset("kernel_bootstrap")
    global rbf_params, poly_params, poly_alpha, rbf_alpha, x
    ns = [30, 300]
    for n in ns:
        x = np.random.random(n)
        y = f_true(x) + np.random.randn(n)

        poly_alpha, poly_params = train(x, y, kernel='poly', fold_num=int(n / 30))
        rbf_alpha, rbf_params = train(x, y, kernel='rbf', fold_num=int(n / 30))

        print("n: {}".format(n))
        print("Poly d: {}\t lambda: {}".format(poly_params[0], poly_params[1]))
        print("RBF gamma: {}\t lambda: {}".format(rbf_params[0], rbf_params[1]))

        x_plot = np.linspace(0, 1, 100)
        poly_y = k_poly(x, x_plot, poly_params[0])
        poly_y = np.matmul(poly_alpha, poly_y)
        rbf_y = k_rbf(x, x_plot, rbf_params[0])
        rbf_y = np.matmul(rbf_alpha, rbf_y)

    plt.plot(X[np.argsort(X)], Y[np.argsort(X)], 'x', label='Original Data')
    plt.plot(x, f_true, label='True $f(x)$')
    plt.plot(x, f_hat, label='$\widehat f(x)$')
    plt.legend()
    plt.savefig('A3b_poly.png')
    plt.show()

    plt.plot(X[np.argsort(X)], Y[np.argsort(X)], 'x', label='Original Data')
    plt.plot(x, f_true, label='True $f(x)$')
    plt.plot(x, f_hat,  label='$\widehat f(x)$')
    plt.legend()
    plt.savefig('A3b_rbf.png')

    plt.figure(figsize=(6.4, 4.8))
    plt.plot(X[np.argsort(X)], Y[np.argsort(X)], 'x', label='Original Data')
    plt.plot(x, f_true, label='True $f(x)$')
    plt.plot(x, f_hat, label='$\widehat f(x)$')
    plt.title("Ploy")
    plt.legend()
    plt.savefig('A3c_poly.png')

    plt.figure(figsize=(6.4, 4.8))
    plt.plot(X[np.argsort(X)], Y[np.argsort(X)], 'x', label='Original Data')
    plt.plot(x, f_true, label='True $f(x)$')
    plt.plot(x, f_hat, label='$\widehat f(x)$')
    plt.title("RBF")
    plt.legend()
    plt.savefig('A3c_rbf.png')

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

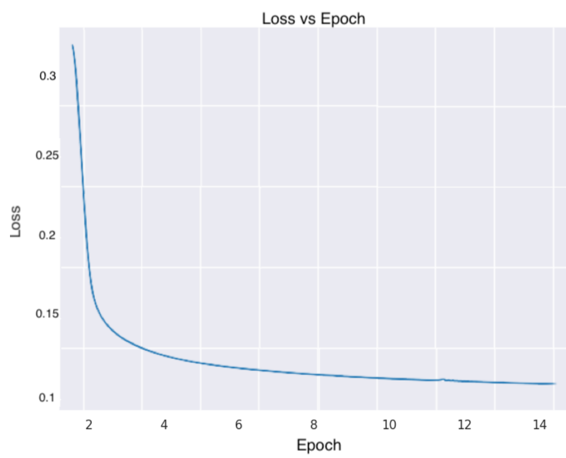
A6:

- **Part a:**



  Loss: 0.1024
  Accuracy: 0.9703

- **Part b:**



  Loss: 0.1262
  Accuracy: 0.9659

- **Part c:**
  Number of parameters for a: 50890
  Number of parameters for b: 26506
  The difference between two accuracy and loss are close. The parameters for part b is half of the part a, which makes F2 more efficient. Also, the deeper network is better, as it can process the complex data very efficient.