Report: Practice #2

ITE4053, **Deep Learning Methods and Applications.**
2016025305, **Jihun Kim<jihunkim@hanyang.ac.kr>.**

Logistic regression과 cross-entropy loss를 이용하는 binary classifier를 구현한다.
구현은 Python과 NumPy를 이용했다.

Implementation

Development Environment

* macOS 10.15.3
* Python 3.6.8
* NumPy 1.17.2

Code

`BinaryClassifier` 클래스를 만들어 구현했다.
각 메소드의 역할은 다음과 같다.

`generate_data(size)`
: `size` 크기의 데이터를 생성한다.

`__forward__(self, X)`
: 주어진 데이터 `X`로 forward pass를 수행한다.

`__backward__(size, X, y)`
: 주어진 데이터 `X`와 레이블 `y`로 backward pass를 수행한다.

`train(self, X, y, learning_rate)`
: forward pass와 backward pass를 순차적으로 진행하여 모델을 학습시킨다.

`predict(self, X)`
: 모델의 예측 결과를 내놓는다. `__forward__`와의 유일한 차이점은 0 또는 1을 결과로 내놓는다는 점이다.

`loss(self, X, y)`
: forward pass를 한 번 진행한 뒤 `y`와 비교해 loss를 계산한다.

```python
class BinaryClassifier:

    def __init__(self):
      pass

    @staticmethod
    def generate_data(size):
      pass

    def __forward__(self, X):
      pass

    def __backward__(self, X, y):
      pass

    def train(self, X, y, learning_rate):
      pass

    def predict(self, X):
      pass

    def loss(self, X, y):
      pass
```

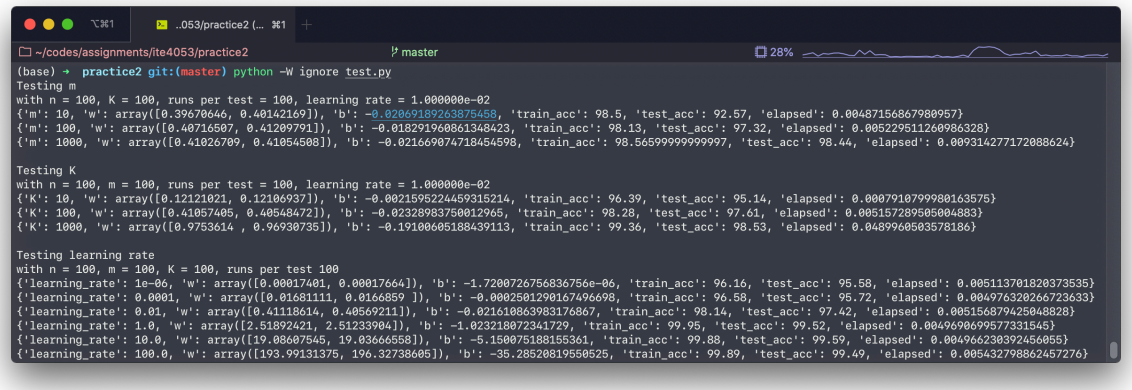
전체 코드는 보고서의 끝에 첨부했다.

Experiments

구현한 코드를 바탕으로 결과를 실험적으로 분석한다.

매 실행마다 결과에 약간의 차이가 있을 수 있으므로, 보다 정확한 테스트를 위해 테스트 코드를 작성했다.
테스트 코드는 100번의 시도들을 통해 얻은 결과들의 평균을 계산한다.

아래 결과들에서, m은 테스트 데이터의 수, K는 학습 횟수를 나타낸다. α는 learning rate를 나타낸다.



Time Comparison

연산 과정을 element-wise로 구현한 버전과 vectorized로 구현한 버전의 학습 속도를 비교한다.

	Time elapsed	Relative time elapsed
Element-wise	203.4ms	20.9
Vectorized	9.7ms	1

$m = 1000,\ K = 100$

vectorized 버전이 무려 20배 이상 빠른 속도를 보여줬다.

Choosing α

$\alpha =$	10^{-6}	10^{-4}	10^{-2}	10^{0}	10^{1}	10^{2}
Train accuracy	96.16%	96.58%	98.14%	99.95%	99.88%	99.89%
Test accuracy	95.58%	95.72%	97.42%	99.52%	99.59%	99.49%
Time elapsed	5.1ms	4.9ms	5.1ms	4.9ms	4.9ms	5.4ms

$m = 100,\ K = 100$

이 예제에서는 learning rate가 충분히 클 때 좋은 성능을 내는 것을 확인할 수 있었다.

Hyperparameter Search

Choosing m, K

$m =$	10	100	1,000
Train accuracy	98.60%	98.06%	98.59%
Test accuracy	93.15%	97.39%	98.47%
Time elapsed	5.4ms	5.3ms	9.7ms

$K = 100, \alpha = 10^{-2}$

$K =$	10	100	1,000
Train accuracy	96.44%	98.04%	99.34%
Test accuracy	95.41%	97.73%	98.54%
Time elapsed	0.7ms	5.4ms	50.1ms

$m = 100, \alpha = 10^{-2}$

학습 데이터의 수보다는 학습 횟수에 따른 성능 향상 폭이 더 컸다.
그러나 학습에 소요되는 시간이 학습 횟수에 거의 비례하여 증가하는 모습을 보였다.

Best Result

가장 좋은 결과를 낸 parameter 조합은 다음과 같다.
$m = 1000, n = 100, K = 1000, \alpha = 10^{0}$

그리고 이 때의 결과는 다음과 같다.
$w_1 = 5.02454678, w_2 = 5.0249903, b = -2.391448015506648$
Train accuracy: 100.0%
Test accuracy: 100.0%
Time elapsed: 91ms

Appendix A: `binary_classifier.py`

```python
import numpy as np
import time

DIM_X = 2  # Dimension of data


class BinaryClassifier:

    def __init__(self):
        self.w = np.zeros((1, DIM_X), dtype=np.float64)  # (1,D)
        self.b = 0

    @staticmethod
    def generate_data(size):
        X = np.random.randint(-10, 11, (size, DIM_X))   # (N,D)
        y = (np.sum(X, 1) > 0).astype(int)  # (N,)
        return X, y

    def __forward__(self, X):
        self.z = np.dot(self.w, X.T) + self.b  # (1,N)
        self.a = 1 / (1 + np.exp(-self.z))  # (1,N)
        MIN_MARGIN = 2 ** -53
        self.a = np.maximum(MIN_MARGIN, np.minimum(1 - MIN_MARGIN, self.a))
        return self.a

    def __backward__(self, X, y):
        da = -y / self.a + (1 - y) / (1 - self.a)  # (1,N)
        dz = self.a * (1 - self.a) * da  # (1,N)
        dw = np.mean(X * dz.T, 0)  # (D,)
        db = np.mean(1 * dz)
        return dw, db

    def train(self, X, y, learning_rate):
        self.__forward__(X)
        dw, db = self.__backward__(X, y)
        self.w -= learning_rate * dw
        self.b -= learning_rate * db

    def predict(self, X):
        return np.round(self.__forward__(X))

    def loss(self, X, y):
        pred_y = self.__forward__(X)
        return -np.mean(y * np.log(pred_y) + (1 - y) * np.log(1 - pred_y))


def train_binary_classifier(num_train, num_test, num_iter, learn_rate):
    classifier = BinaryClassifier()
    train_X, train_y = classifier.generate_data(num_train)
    test_X, test_y = classifier.generate_data(num_test)
    for iteration in range(num_iter):
        classifier.train(train_X, train_y, learn_rate)
    return {'w': classifier.w.reshape(DIM_X), 'b': classifier.b,
            'train_loss': classifier.loss(train_X, train_y),
            'test_loss': classifier.loss(test_X, test_y),
            'train_acc': 100 * np.mean(classifier.predict(train_X) == train_y),
            'test_acc': 100 * np.mean(classifier.predict(test_X) == test_y)}


if __name__ == '__main__':
    NUM_TRAIN = 1000  # Number of train data
    NUM_TEST = 100  # Number of test data
    LEARN_RATE = 1e-2  # Learning rate
    NUM_ITER = 1000  # Number of iterations

    classifier = BinaryClassifier()
    train_X, train_y = classifier.generate_data(NUM_TRAIN)
    test_X, test_y = classifier.generate_data(NUM_TEST)
    start = time.time()
    for iteration in range(NUM_ITER + 1):
        if iteration:
            classifier.train(train_X, train_y, LEARN_RATE)
        print('===== Iteration #' + str(iteration) + " =====")
        for i in range(classifier.w.shape[1]):
            print('w' + str(i + 1) + ' = ' + str(classifier.w[0][i]))
        print('b = ' + str(classifier.b))
        print('Train loss = ' + str(classifier.loss(train_X, train_y)))
        print('Test loss = ' + str(classifier.loss(test_X, test_y)))
        print('Train accuracy = ' + str(100 * np.mean(classifier.predict(train_X) == train_y)) + '%')
        print('Test accuracy = ' + str(100 * np.mean(classifier.predict(test_X) == test_y)) + '%')
        print()
    end = time.time()
    print('Time elapsed: ' + str(end - start) + 's')
```

Appendix B: `test.py`

```python
from binary_classifier import *
import time


def run_test(m, n, K, learning_rate, num_runs):
    test_result = {'w': np.zeros(DIM_X), 'b': 0, 'train_acc': 0, 'test_acc': 0, 'elapsed': -time.time()}
    for run in range(num_runs):
        cls_result = train_binary_classifier(m, n, K, learning_rate)
        test_result['train_acc'] += cls_result['train_acc']
        test_result['test_acc'] += cls_result['test_acc']
        test_result['w'] += cls_result['w']
        test_result['b'] += cls_result['b']
    test_result['elapsed'] += time.time()
    for key, val in test_result.items():
        test_result[key] = val / num_runs
    return test_result


def test_m(m_list, n, K, learning_rate, num_runs):
    test_results = []
    for m in m_list:
        cur_result = {'m': m}
        cur_result.update(run_test(m, n, K, learning_rate, num_runs))
        test_results.append(cur_result)
    return test_results


def test_K(m, n, K_list, learning_rate, num_runs):
    test_results = []
    for K in K_list:
        cur_result = {'K': K}
        cur_result.update(run_test(m, n, K, learning_rate, num_runs))
        test_results.append(cur_result)
    return test_results


def test_lr(m, n, K, lr_list, num_runs):
    test_results = []
    for lr in lr_list:
        cur_result = {'learning_rate': lr}
        cur_result.update(run_test(m, n, K, lr, num_runs))
        test_results.append(cur_result)
    return test_results


if __name__ == '__main__':

    m = 100
    n = 100
    K = 100
    num_run = 100
    lr = 1e-2

    # Test m
    print('Testing m')
    print('with n = %d, K = %d, runs per test = %d, learning rate = %e' % (n, K, num_run, lr))
    test_results = test_m([10, 100, 1000], n, K, lr, num_run)
    for i in test_results:
        print(i)
    print()

    # Test K
    print('Testing K')
    print('with n = %d, m = %d, runs per test = %d, learning rate = %e' % (n, m, num_run, lr))
    test_results = test_K(m, n, [10, 100, 1000], lr, num_run)
    for i in test_results:
        print(i)
    print()

    # Test learning rate
    print('Testing learning rate')
    print('with n = %d, m = %d, K = %d, runs per test %d' % (n, m, K, num_run))
    test_results = test_lr(m, n, K, [1e-6, 1e-4, 1e-2, 1e0, 1e1, 1e2], num_run)
    for i in test_results:
        print(i)
    print()

    print(run_test(1000, 100, 1000, 1, 100))
```