Lab 5: Regression & Neural Networks

519021910702 游克垚

1 Exercise 1: Linear Regression

1.1 Implemention

My implemention is as follows:

```
import numpy as np

if __name__ == "__main__":
    X = np.load('Data1_X.npy')
    Y = np.load('Data1_Y.npy')

    X = np.matrix(X)
    Y = np.matrix(Y)

    Y_hat = X * (X.T * X).I * X.T * Y
```

1.2 Result

The result of $\hat{Y}^T$ is as follows:

```
[[ 90.19765634  47.67686144  91.65938765 120.74673813 115.32784698
   99.5235342  101.6811349   73.64256436 118.02996154 101.22977512
   53.90993768  80.72800342 100.06768951  84.87342942 108.10549645
   84.28833524  87.50086209  87.23821599  82.31093561 104.35109097
   69.9964474   97.5290999  113.73176903  83.13355445 125.47193094
   91.16712382  87.03894768  94.06534565  72.53264476 105.78014471
  109.47621827  93.9088358  110.22388501  83.04389589 105.0040709
   64.34001661  58.32922243  90.39147089 113.32483653  62.51171231
   86.12664693  78.76219229  58.93439827  42.12567826 126.62571301
  106.33033386  46.80686218  64.43015537  60.12018271  58.64341279
  103.05922202 108.34444014  59.46080212 136.30383353  86.79373439
   81.59990756  91.86133364 100.73337199  86.9771913   70.95011886
   75.72726683  97.88306054  34.9951482  100.82833137 109.88897963
   92.81345368  89.46905032 136.37900461  65.90720482  82.27029322
  111.67846566  46.88349888  69.84761845  95.91483137  62.0903558
   74.21588583  79.1009826   71.19647357 127.98576243  59.55125149
```

```
   94.22580938  82.5433742   74.60546716  87.18355727  99.21624982
   91.55587898 114.7186236  107.22598751  89.13747846 112.2219674
   91.2314803  104.22842129 102.5151316  109.87026128 117.48132199
   78.98080158  50.16010039  85.87762567  85.88482494 106.62073286]]
```

Under the geometric interpretation of linear regression, our goal is to find a point $\hat{Y}$ in the X plane which is the nearest to the point Y. So vector $Y - \hat{Y}$ must be perpendicular to the X plane. So

$$X^T(Y - \hat{Y}) = 0$$
$$X^T(Y - X\hat{\theta}) = 0$$
$$\hat{\theta} = (X^TX)^{-1}X^TY$$

So $\hat{Y} = X\hat{\theta} = X(X^TX)^{-1}X^TY$.

2 Exercise 2: Logistic Regression

2.1 Implemention

My implemention is as follows:

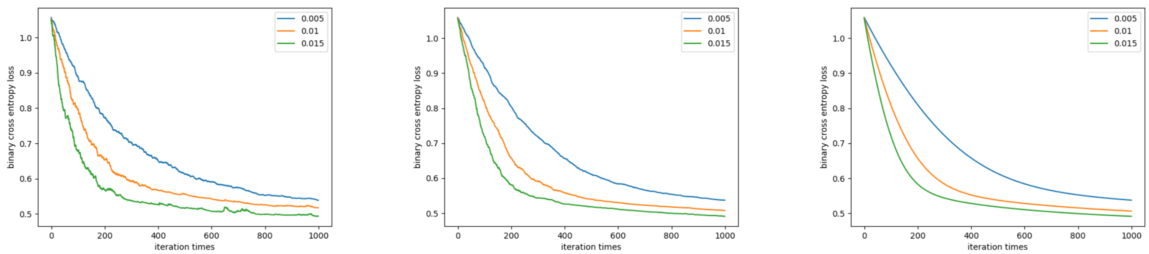
```python
import numpy as np
import matplotlib.pyplot as plt
from utils import plot_decision_boundary

def CELoss_binary(X, y, theta):
    z = X @ theta
    y_hat = 1 / (1 + np.exp(-z))
    return - (y.T @ np.log(y_hat + 0.000001) + (1-y).T @ np.log(1-y_hat + 0.000001)) / X.shape[0]

def precision(y_true, y_predict):
    return np.sum(y_true == y_predict) / y_true.shape[0]

def predict(X, theta, threshold):
    z = X @ theta
    y_hat = 1 / (1+np.exp(-z))
    y_predict = np.zeros(y_hat.shape)
    y_predict[y_hat > threshold] = 1
    return y_predict

def gradient(X, y, theta):
    z = X @ theta
    y_hat = 1 / (1 + np.exp(-z))
    return X.T @ (y_hat-y) / X.shape[0]

if __name__ == "__main__":
```

```
    X_all, y_all = np.load('Data2_X.npy'), np.load('Data2_Y.npy')

    b = np.ones((X_all.shape[0], 1))
    X_all = np.c_[X_all, b]

    for k in range(3):
        theta = np.ones((X_all.shape[1],))

        lr = 0.01
        iteration_num = 1000
        threshold = 0.2 + k * 0.3

        loss = []
        prec = []

        for i in range(iteration_num):
            # change algorithm
            index = np.random.choice(np.arange(X_all.shape[0]), size=100, replace=False)
            X = X_all[index]
            y = y_all[index]
            theta -= lr * gradient(X, y, theta)

            y_predict = predict(X_all, theta, threshold)

            loss.append(CELoss_binary(X_all, y_all, theta))
            prec.append(precision(y_all, y_predict))

        # plt.plot([i for i in range(iteration_num)], loss, label=str(threshold))
        # plt.plot([i for i in range(iteration_num)], prec, label=str(threshold))

        plot_decision_boundary(X_all, y_all, lambda x : predict(np.c_[x, np.ones((x.shape[0], 1))], theta, threshold))
        plt.show()

    # plt.legend()

    # plt.xlabel('iteration times')
    # plt.ylabel('precision')
    # plt.ylabel('binary cross entropy loss')

    # plt.show()
```

When counting CELoss_binary, I add a small number to $\hat{y}$ to avoid $np.log(0)$.

2.2 Binary cross entropy loss

The graphs of binary cross entropy loss against the number of iterations using stochastic gradient descent, mini-batch gradient descent and (batch) gradient descent respectively under 3 different learning rates are as follows:

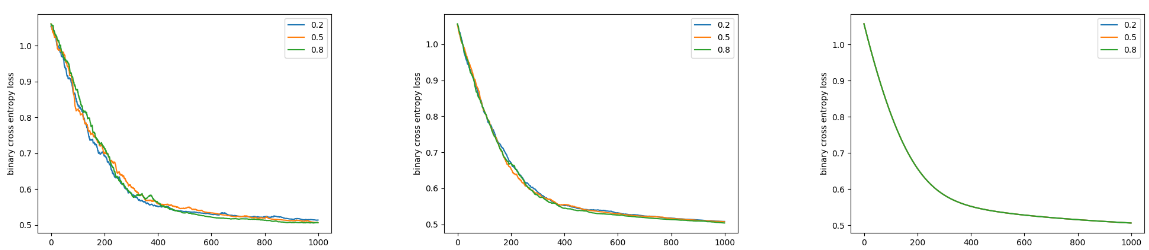


(a) stochastic gradient descent (b) mini-batch gradient descent (c) batch gradient descent

Fig 2-1 binary cross entropy loss under 3 different learning rates

If the learning rate is larger, the binary cross entropy loss descends quicker. And the binary cross entropy loss fluctuates most when using stochastic gradient descent, less when using mini-batch gradient descent and least when using batch gradient descent. The stochastic gradient descent is also the fastest, followed by mini-batch gradient descent, and the batch gradient descent is the slowest.

The graphs of binary cross entropy loss against the number of iterations using stochastic gradient descent, mini-batch gradient descent and (batch) gradient descent respectively under 3 different values of the threshold are as follows:



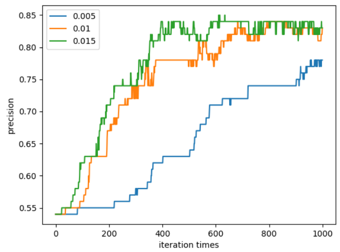
(a) stochastic gradient descent (b) mini-batch gradient descent (c) batch gradient descent

Fig 2-2 binary cross entropy loss under 3 different values of the threshold

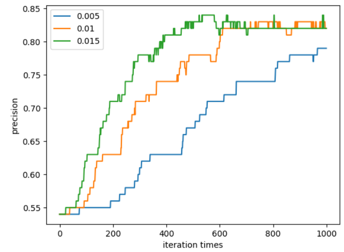
The value of threshold doesn't effect binary cross entropy loss much because it only works when predicting. The effects of types of gradients are the same.

2.3 Precision

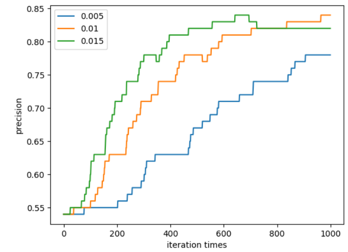
The graphs of precision against the number of iterations using stochastic gradient descent, mini-batch gradient descent and (batch) gradient descent respectively under 3 different learning rates are as follows:



(a) stochastic gradient descent



(b) mini-batch gradient descent

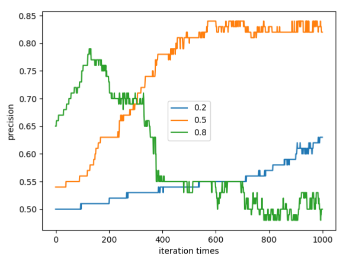


(c) batch gradient descent

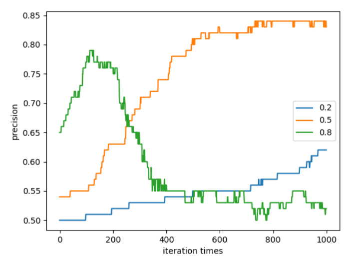
Fig 2-3 precision under 3 different learning rates

Similar to the graphs above, the learning rate influences the speed of learning. But when the learning rate is too large, it is difficult to converge and the precision is low. The effects of types of gradients are the same.

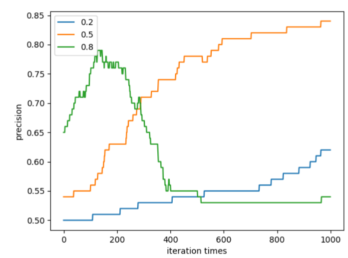
The graphs of precision under 3 different values of the threshold are as follows:



(a) stochastic gradient descent



(b) mini-batch gradient descent



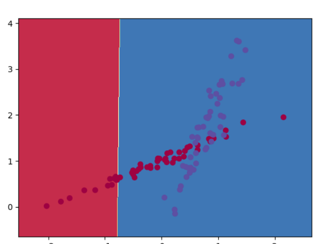
(c) batch gradient descent

Fig 2-4 precision under 3 different values of the threshold

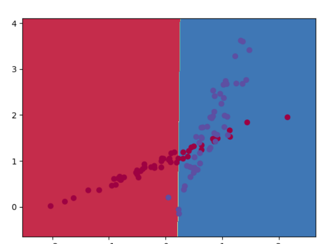
The value of the threshold is important to precision because it decides the decision boundary. The effects of types of gradients are the same.

2.4 Decision boundary

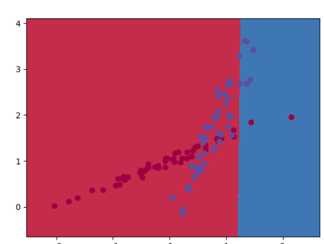
The graphs of decision boundary of predictions with different values of threshold are as follows:



(a) threshold=0.2



(b) threshold=0.5



(c) threshold=0.8

Fig 2-5 decision boundary of predictions with different values of threshold

We can find that if the value of threshold is smaller, the decision boundary moves more in the negative direction of the x-axis. But with the number of iteration increasing, they will gradually approach to each other.

3 Exercise 3: L1/L2 Regularization

3.1 Ridge regression

My implemention is as follows:

```python
def ridge_regression(x, y, lamda):
    # Normalize data.
    x = (x - x.mean(axis=-1, keepdims=True)) / x.std(axis=-1, keepdims=True)
    from sklearn.linear_model import Ridge
    ridge = Ridge(alpha=lamda)
    ridge.fit(x, y)
    y_pred = ridge.predict(x)  # predicted labels of size (n_samples, )
    intercept = ridge.intercept_  # b of size ()
    coef = ridge.coef_  # theta of size (n_dims, )
    return y_pred, intercept, coef
```

The result of ridge regression is as follows:

```
0.03876948356628418 1.042289124220179 0.0
0.0012583732604980469 1.1052304919020004 0.0
0.0005276203155517578 1.5737740179631936 0.0
0.0006253719329833984 1.6765680529544285 0.0
0.000644683837890625 1.732565285766469 0.0
0.0005466938018798828 7.331330534314156 0.0
```

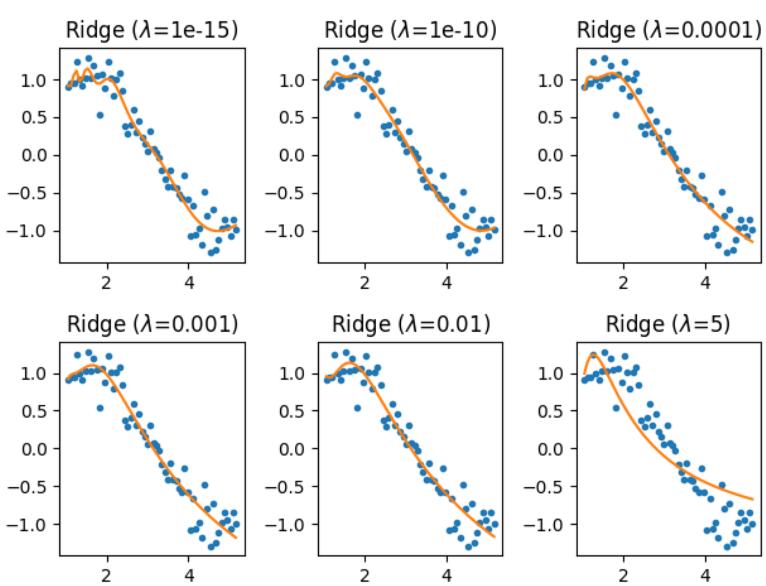


Fig 3-6 result of ridge regression

3.2 Lasso regression

My implemention is as follows:

```python
def lasso_regression(x, y, lamda):
    # Normalize data.
    x = (x - x.mean(axis=-1, keepdims=True)) / x.std(axis=-1, keepdims=True)

    from sklearn.linear_model import Lasso

    lasso = Lasso(alpha=lamda, max_iter=1000000)
    lasso.fit(x, y)

    y_pred = lasso.predict(x)  # predicted labels of size (n_samples, )
    intercept = lasso.intercept_  # b of size ()
    coef = lasso.coef_  # theta of size (n_dims, )

    return y_pred, intercept, coef
```

The result of lasso regression is as follows:

```
1.448115348815918 1.333009473823675 0.0
0.4232501983642578 1.5347481924504889 68.75
0.003134012222290039 1.7176378773851337 75.0
0.0009174346923828125 1.8505307867417076 81.25
0.0005638599395751953 3.134818651748969 87.5
0.000553131103515625 40.4165451776714 100.0
```

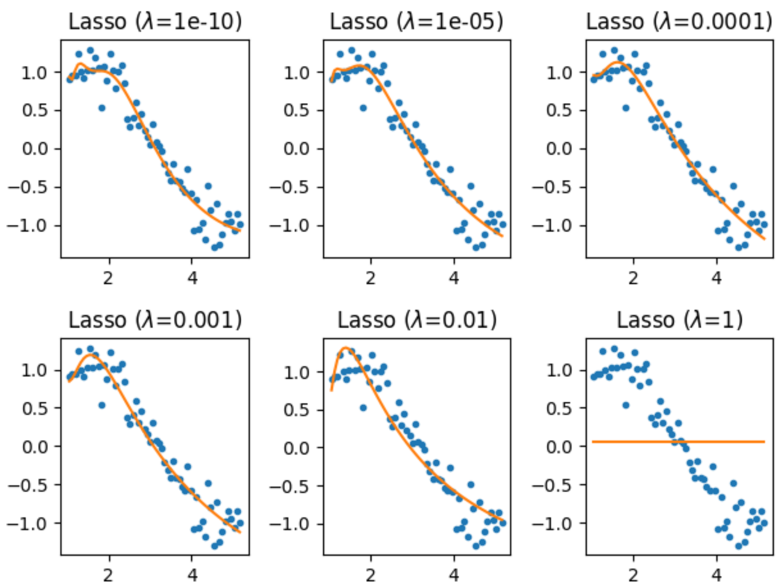


Fig 3-7 result of lasso regression

3.3 Discussion

We can find ridge regression spends less time while lasso regression spends more time. So ridge regression is less computationally expensive. And lasso regression tends to create a sparser output. Generalization refers a model adjusting itself based on the training data. Regularization refers a model reducing coefficients in learning and preventing overfitting. So regularization is a method to increase generalization.

4 Exercise 4: Two-layer Perceptron Network

4.1 Implemention

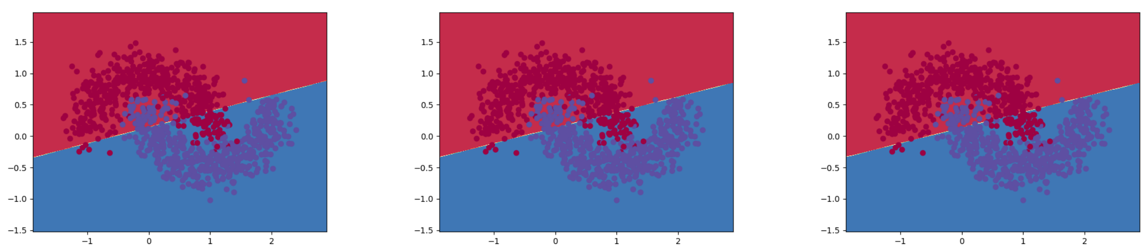
My implemention is as follows:

```
class Relu(Layer):
    def __init__(self, name):
        super(Relu, self).__init__(name)

    def forward(self, input):
        self._saved_for_backward(input)
        input[input <= 0] = 0

        return input

    def backward(self, grad_output):
        grad_output[self._saved_tensor <= 0] = 0

        return grad_output

class Linear(Layer):
    def __init__(self, name, in_num, out_num, init_std):
        super(Linear, self).__init__(name, trainable=True)

        self.in_num = in_num
        self.out_num = out_num
        self.W = np.random.randn(in_num, out_num) * init_std
        self.b = np.zeros(out_num)

        self.grad_W = np.zeros((in_num, out_num))
        self.grad_b = np.zeros(out_num)

        self.diff_W = np.zeros((in_num, out_num))
        self.diff_b = np.zeros(out_num)

    def forward(self, input):
        self._saved_for_backward(input)

```

```python
        return input.dot(self.W) + self.b

    def backward(self, grad_output):
        self.grad_W = self._saved_tensor.T @ grad_output
        self.grad_b = grad_output.sum(axis=0)

        return grad_output @ self.W.T

    def update(self, config):
        mm = config['momentum']
        lr = config['learning_rate']

        self.W -= lr * self.grad_W
        self.b -= lr * self.grad_b

class Network(object):
    def __init__(self):
        self.layer_list = []
        self.params = []
        self.num_layers = 0

    def add(self, layer):
        self.layer_list.append(layer)
        self.num_layers += 1

    def forward(self, input):
        output = input
        for i in range(self.num_layers):
            output = self.layer_list[i].forward(output)

        return output

    def backward(self, grad_output):
        grad = grad_output
        for i in range(self.num_layers-1, -1, -1):
            grad = self.layer_list[i].backward(grad)

        return grad

    def update(self, config):
        for i in range(self.num_layers):
            if self.layer_list[i].trainable:
                self.layer_list[i].update(config)

    def predict(self, input):
        y_pred = self.forward(input).argmax(axis=-1)

        return y_pred
```

```python

class EuclideanLoss:
    def __init__(self, name):
        self.name = name

    def forward(self, input, target):
        return ((target - input) ** 2).mean(axis=0).sum() / 2.

    def backward(self, input, target):
        return (input - target)
```

4.2 Result



(a) hidden_dim=2 (b) hidden_dim=50 (c) hidden_dim=200

Fig 4-8 result of two-layer perceptron network

The result of two-layer perceptron network having different numbers of hidden neurons is the same. But if the number of hidden neurons is smaller, it is more likely to produce a failure as below.

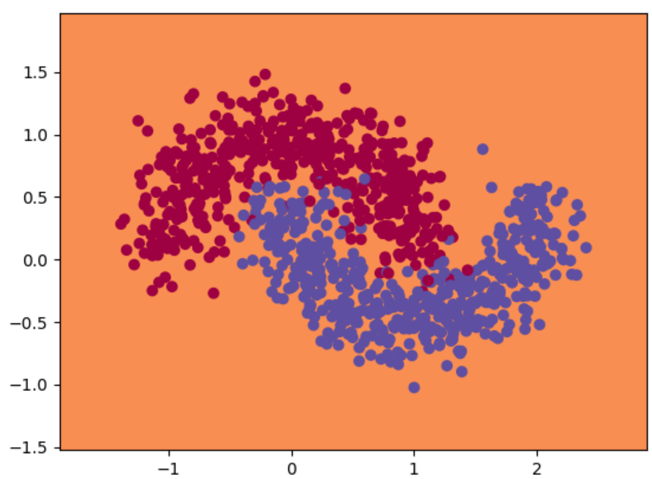


Fig 4-9 a failure of training