ass2_grad

April 1, 2018

```
In [1]: import numpy as np
        import matplotlib.pyplot as plt
        from data_util_KNN import readMnist

In [2]: X_train, y_train, X_test, y_test = readMnist('E:\Code\jupyterpy\Digital'
                                                      'Video\datasets\mnist')

In [3]: mask = range(5000)
        X_tr = X_train[mask]
        y_tr = y_train[mask]
        mask = range(500)
        X_te = X_test[mask]
        y_te = y_test[mask]

In [4]: W = np.random.random((X_tr.shape[1] + 1, 10))
        print(W.shape)
        W

(785, 10)


Out[4]: array([[ 0.80494852,  0.90765804,  0.72454771, ...,  0.30158744,
                 0.90272307,  0.6233478 ],
               [ 0.91114889,  0.84165155,  0.70317371, ...,  0.5170203 ,
                 0.56772921,  0.09373554],
               [ 0.66437615,  0.66651792,  0.61080153, ...,  0.56486989,
                 0.79127343,  0.17587681],
               ...,
               [ 0.45672482,  0.14032603,  0.73310886, ...,  0.57979093,
                 0.28008893,  0.80227858],
               [ 0.76021803,  0.60730959,  0.32624087, ...,  0.57817323,
                 0.29905183,  0.71498281],
               [ 0.62510794,  0.48628265,  0.45963862, ...,  0.56807162,
                 0.22437052,  0.32629691]])

In [5]: def featurenormal(X):
            std = np.std(X, axis=1).reshape(-1, 1)
            mean = np.mean(X, axis=1).reshape(-1, 1)
```

```
        X_change = (X - mean) / std
        return X_change, mean, std
    X_tr, mean, std= featurenormal(X_tr)
```

#softmaxLoss

$$L_i = -log\frac{e^{s_k}}{\sum_j e^{s_j}}$$

$$L_i = -\sum_k P_{ik} log P_k \qquad \frac{\partial L_i}{\partial P_k} = -\frac{1}{P_k}$$

$$P_k = \frac{e^{s_k}}{\sum_j e^{s_j}}$$

$$k = m: \quad \frac{\partial P_k}{\partial s_m} = -\frac{e^{s_m}\sum_j e^{s_j} - (e^{s_m})^2}{(\sum_j e^{s_j})^2} = P_m(1 - P_m) \quad k \neq m: \quad \frac{\partial P_k}{\partial s_m} = -\frac{e^{s_k}e^{s_m}}{(\sum_j e^{s_j})^2} = -P_k P_m$$

$$s_m = (X_i \times W)_m \qquad \frac{\partial s_m}{\partial W} = X_i$$

$$k = m: \quad \frac{\partial L_i}{\partial W} = P_m - 1 \quad k \neq m: \quad \frac{\partial L_i}{\partial W} = P_m$$

```
In [6]: def computeSoftmaxLoss(X_tr, y_tr, W, reg):
            n, w = X_tr.shape
            X_tr = np.c_[np.ones((n, 1)), X_tr]
            s_mat = X_tr.dot(W)
            s_m_argmax = np.argmax(s_mat, axis=1)
            s_m_max = s_mat[range(n), s_m_argmax].reshape(-1, 1)
            s_m_exp = np.exp(s_mat - s_m_max)
            L_mat = - np.log(s_m_exp[range(n), y_tr] / np.sum(s_m_exp, axis=1))
            L = np.sum(L_mat) / n
            L += 0.5 * reg * np.sum(W ** 2)
            dW = np.zeros(W.shape)

            '''
            for i in range(n):
                for m in range(W.shape[1]):
                    P_m = np.reshape((s_m_exp[range(n), m] /
                                        np.sum(s_m_exp, axis=1)), (-1, 1))
                    if (m == y_tr[i]):
                        dW[:, m] += X_tr[i] * (P_m[i] - 1)
                    else:
                        dW[:, m] += X_tr[i] * (P_m[i])
            '''

            P_m = (s_m_exp / np.sum(s_m_exp, axis=1).reshape(-1, 1))
            P_ik = np.zeros(P_m.shape)
            P_ik[range(n), y_tr] = 1
```

```
        dW = X_tr.T.dot(P_m - P_ik)
        dW /= n
        dW += reg * W
        return L, dW
```

```
In [7]: loss_softmax, grad_softmax = computeSoftmaxLoss(X_tr[:10], y_tr[:10],
                                                         W, reg=0)
```

```
In [8]: def numComputeSoftmax(X_tr, y_tr, W, reg, h=0.000001):
            w, k = W.shape
            dW = np.zeros((w, k))
            for i in range(w):
                for j in range(k):
                    loss = computeSoftmaxLoss(X_tr[:10], y_tr[:10], W, 0)[0]
                    W[i, j] += h
                    loss_c = computeSoftmaxLoss(X_tr[:10], y_tr[:10], W, 0)[0]
                    dW[i, j] = (loss_c - loss) /  h
            return dW
```

```
In [9]: num_softmax = numComputeSoftmax(X_tr[:10], y_tr[:10], W, 0)
        num_softmax
```

```
Out[9]: array([[ -2.85524191e-02,  -2.93491350e-01,   9.50748813e-04, ...,
                  1.45805453e-01,   3.43865381e-04,  -9.94415448e-02],
               [  1.57605609e-02,   9.00941028e-02,   1.29820314e-02, ...,
                 -6.60884787e-02,  -1.24215305e-04,   3.96309474e-02],
               [  1.57605626e-02,   9.00941011e-02,   1.29820314e-02, ...,
                 -6.60884805e-02,  -1.24215305e-04,   3.96309456e-02],
               ...,
               [  1.57605644e-02,   9.00941011e-02,   1.29820314e-02, ...,
                 -6.60884787e-02,  -1.24215305e-04,   3.96309456e-02],
               [  1.57605609e-02,   9.00941046e-02,   1.29820314e-02, ...,
                 -6.60884787e-02,  -1.24215305e-04,   3.96309456e-02],
               [  1.57605626e-02,   9.00941046e-02,   1.29820297e-02, ...,
                 -6.60884787e-02,  -1.24215305e-04,   3.96309456e-02]])
```

```
In [10]: grad_softmax
```

```
Out[10]: array([[ -2.85524313e-02,  -2.93491353e-01,   9.50748654e-04, ...,
                   1.45805473e-01,   3.43866960e-04,  -9.94415454e-02],
                [  1.57605607e-02,   9.00941018e-02,   1.29820311e-02, ...,
                  -6.60884772e-02,  -1.24215382e-04,   3.96309468e-02],
                [  1.57605607e-02,   9.00941018e-02,   1.29820311e-02, ...,
                  -6.60884772e-02,  -1.24215382e-04,   3.96309468e-02],
                ...,
                [  1.57605607e-02,   9.00941018e-02,   1.29820311e-02, ...,
                  -6.60884772e-02,  -1.24215382e-04,   3.96309468e-02],
                [  1.57605607e-02,   9.00941018e-02,   1.29820311e-02, ...,
                  -6.60884772e-02,  -1.24215382e-04,   3.96309468e-02],
```

```
[ 1.57605607e-02,   9.00941018e-02,   1.29820311e-02, ...,
 -6.60884772e-02,  -1.24215382e-04,   3.96309468e-02]])
```

```
In [11]: diff = np.linalg.norm(num_softmax - grad_softmax) / \
                        np.linalg.norm(num_softmax + grad_softmax)
         diff
```

```
Out[11]: 8.9204458062729595e-08
```

##HingeLoss

$$L_i = \sum_{j \neq y_i} max(0, s_j - s_{y_i} + 1)$$

$$k = m: \quad L_i = \sum_{j \neq k} max(0, s_j - s_k + 1) \quad \frac{\partial L_i}{\partial s_m} = \sum_{j \neq k} -1 \quad if: (s_j - s_k + 1) > 0$$

$$k \neq m: \quad L_i = \sum_{j \neq k} max(0, s_j - s_k + 1) \quad \frac{\partial L_i}{\partial s_m} = \sum_{j \neq k} 1 \quad if: (s_j - s_k + 1) > 0$$

$$s_m = (X_i \times W)_m \qquad \frac{\partial s_m}{\partial W} = X_i$$

$$k = m: \quad \frac{\partial L_i}{\partial W} = \sum_{j \neq k} -X_i \quad if: (s_j - s_k + 1) > 0 \qquad k \neq m: \quad \frac{\partial L_i}{\partial W} = \sum_{j \neq k} X_i \quad if: (s_j - s_k + 1) > 0$$

```
In [12]: def computeHingeLoss(X_tr, y_tr, W, reg):
             n, w = X_tr.shape
             X_tr = np.c_[np.ones((n, 1)), X_tr]
             s_mat = X_tr.dot(W)
             s_k = s_mat[range(n), y_tr]
             pred_score = s_mat - s_k.reshape(-1, 1) + 1
             pred_score[range(n), y_tr] = 0
             L = np.sum(pred_score[np.where(pred_score > 0)]) / n
             L += 0.5 * reg * np.sum(W ** 2)
             dW = np.zeros(W.shape)
             pred_score[pred_score < 0] = 0
             '''
             for i in range(n):
                 for m in range(W.shape[1]):
                     if pred_score[i, m] > 0:
                         dW[:, m] += X_tr[i]
                         dW[:, y_tr[i]] -= X_tr[i]
             '''

             pred_score[pred_score > 0] = 1
             sum_score = np.sum(pred_score, axis=1)
             pred_score[range(n), y_tr] = - sum_score
             dW = X_tr.T.dot(pred_score)
```

```
        dW /= n
        dW += reg * W
        return L, dW
```

```
In [13]: loss_hinge, grad_hinge = computeHingeLoss(X_tr[:10], y_tr[:10], W, 0)
```

```
In [14]: def numHingeCompute(X_tr, y_tr, W, reg, h=0.000001):
             w, k = W.shape
             dW = np.zeros((w, k))
             for i in range(w):
                 for j in range(k):
                     loss = computeHingeLoss(X_tr[:10], y_tr[:10], W, 0)[0]
                     W[i, j] += h
                     loss_c = computeHingeLoss(X_tr[:10], y_tr[:10], W, 0)[0]
                     dW[i, j] = (loss_c - loss) /  h
             return dW
```

```
In [15]: num_hinge = numHingeCompute(X_tr[:10], y_tr[:10], W, 0)
         num_hinge
```

```
Out[15]: array([[-0.1       , -1.10000001, -0.09999999, ...,  0.8       ,
                  0.3       , -0.50000001],
                [ 0.05048132,  0.31159955,  0.0716095 , ..., -0.32712889,
                 -0.11203351,  0.21915552],
                [ 0.05048132,  0.31159955,  0.07160951, ..., -0.32712889,
                 -0.1120335 ,  0.21915551],
                ...,
                [ 0.05048132,  0.31159955,  0.07160951, ..., -0.32712888,
                 -0.11203351,  0.21915552],
                [ 0.05048132,  0.31159955,  0.0716095 , ..., -0.32712888,
                 -0.11203351,  0.21915551],
                [ 0.05048132,  0.31159955,  0.0716095 , ..., -0.32712888,
                 -0.11203351,  0.21915551]])
```

```
In [16]: grad_hinge
```

```
Out[16]: array([[-0.1       , -1.1       , -0.1       , ...,  0.8       ,
                  0.3       , -0.5       ],
                [ 0.05048132,  0.31159955,  0.07160951, ..., -0.32712889,
                 -0.1120335 ,  0.21915551],
                [ 0.05048132,  0.31159955,  0.07160951, ..., -0.32712889,
                 -0.1120335 ,  0.21915551],
                ...,
                [ 0.05048132,  0.31159955,  0.07160951, ..., -0.32712889,
                 -0.1120335 ,  0.21915551],
                [ 0.05048132,  0.31159955,  0.07160951, ..., -0.32712889,
                 -0.1120335 ,  0.21915551],
                [ 0.05048132,  0.31159955,  0.07160951, ..., -0.32712889,
                 -0.1120335 ,  0.21915551]])
```

In [17]:
```
diff = np.linalg.norm(num_hinge - grad_hinge) /  \
                np.linalg.norm(num_hinge + grad_hinge)
diff
```

Out[17]:
```
4.012536292715479e-09
```