计算机科学与技术学院神经网络与深度学习课程实验报告

实验题目： 图像分类与线性分类		**学号：** 201800130086
日期： 2020-10-10	**班级：** 18 智能班	**姓名：** 徐鹏博
Email： hsupengbo@163.com		

实验目的：

- k-Nearest Neighbor classifier
- SVM
- Softmax
- Three-layer-net

实验软件和硬件环境：

Anaconda3,
JupyterNotebook
Inter(R) Core(TM) i5-8250 CPU @1.60GHz 1.80GHz
Win10-x64

实验原理和方法：

KNN:

基本思想是一个样本与数据集中的 k 个样本最相似， 如果这 k 个样本中的大多数属于某一个类别，则该样本也属于这个类别。

使用欧式距离表示两个实例的相似性，$L_p(x_i, x_j) = (\sum_{l=1}^{n} |x_i^{(l)} - x_j^{(l)}|^p)^{\frac{1}{p}}$

SVM:

SVM 的训练目标是寻找一个能将 $y^{(i)}=1$ 和 $y^{(i)}=-1$ 这两类点进行分割的最大间隔超平面。

Loss function 对 w 的偏导如下:

$$L_i = \sum_{j \neq y_i} \left[\max(0, w_j^T x_i - w_{y_i}^T x_i + 1) \right]$$

$$\nabla_{w_{y_i}} L_i = -\left(\sum_{j \neq y_i} 1(w_j^T x_i - w_{y_i}^T x_i + \Delta > 0)\right) x_i$$

$$\nabla_{w_j} L_i = 1(w_j^T x_i - w_{y_i}^T x_i + \Delta > 0) x_i$$

Softmax:

元素的 softmax 值定义为:

$$S_i = \frac{e^i}{\sum_j e^j}$$

使用交叉熵作为损失函数，Loss 计算公式:

$$L_i = \text{-}\log(\frac{e^{f_{y_i}}}{\sum_j e^{f_j}}) = -f_{y_i} + \log\sum_j e^{f_j}$$

$$\frac{\partial loss}{\partial w} = \frac{\partial loss}{\partial scores} \cdot \frac{\partial \text{scores}}{\partial w} \quad \text{Loss} = \text{L}_\text{i}, \text{scores} = \text{f}_\text{j} \quad \frac{\partial f_\text{j}}{\partial w_j} = x_i^T$$

当 $j \neq y_i$ 时, $\frac{\partial L_i}{\partial f_\text{j}} = \left(\text{-}\log(\frac{e^{f_{y_i}}}{\log\sum_j e^{f_j}})\right)' = \frac{e^{f_{y_i}}}{\sum_j e^{f_j}}$

$$\frac{\partial L_i}{\partial w_\text{j}} = \frac{e^{f_{y_i}}}{\sum_j e^{f_j}} \cdot x_i^T$$

当 $j = y_i$ 时, $\frac{\partial L_i}{\partial f_j} = \left(\text{-}\log\left(\frac{e^{f_{y_i}}}{\log\sum_j e^{f_j}}\right)\right)' = \frac{e^{f_{y_i}}}{\sum_j e^{f_j}} - 1$

$$\frac{\partial L_i}{\partial w_\text{j}} = (\frac{e^{f_{y_i}}}{\sum_j e^{f_j}} \text{-} 1) \cdot x_i^T$$

Three-layer-net:

$$1 - layer \ : f_1 = W_1 x + b_1$$

$$2 - layer : f_2 = W_2 \max(0, W_1 x + b_1) + b_2$$

$$3 - layer : f_3 = W_3 \max(0, W_2 \max(0, W_1 x + b_1) + b_2) + b_3$$

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

$$L = \frac{1}{N} \sum_{i=1}^{N} L_i + \lambda \sum_k W_k^2$$

实验步骤：（不要求罗列完整源代码）

KNN:

1. 完成 2 重循环计算测试集和训练集数据的欧式距离:

```
for i in range(num_test):
        for j in range(num_train):
            dists[i][j]=np.sqrt(np.sum(np.power(X[i]-self.X_train[j],2)))
            pass
```

2. 完成 1 重循环计算测试集和训练集数据的欧式距离:

```
for i in range(num_test):
        dists[i, :]=np.sqrt(np.sum(np.power(X[i]-self.X_train,2),axis=1))
        pass
```

3. 完成 0 重循环计算测试集和训练集数据的欧式距离:

直接用矩阵乘，$\sqrt{\|X_i\|^2 + \|Y_j\|^2 - 2 * X_i Y_j'}$

```
    XY=np.multiply(np.dot(X,self.X_train.T),-2)
    X_2=np.sum(self.X_train**2,axis=1)
    Y_2=np.sum(X**2,axis=1,keepdims=True)
    dists=np.add(np.add(np.add(dists,XY),X_2),Y_2)
    dists=np.sqrt(dists)
    pass
```

4. 完成预测函数:

```
for i in range(num_test):
        closest_y = []
        c = np.argsort(dists[i, :])
        k_index = c[:k]
        closest_y = self.y_train[k_index]
        y_pred[i]=Counter(closest_y).most_common(1)[0][0]
        pass
```

5. 完成交叉验证:

采用 S 折交叉验证的方法，即将数据平均分成 S 份，一份作为测试集，其余作为训练集。

表示将数据训练集平均分成 5 份，对应的标签也分为 5 份；

```
X_train_folds=np.array_split(X_train,num_folds)
y_train_folds=np.array_split(y_train,num_folds)

k_to_accuracies = {}
```

依次对每个 k 值，选取一份测试，其余训练，计算准确率，由于之前平均分为 5 份，故对每个 k 都有五个结果

```
for k in k_choices:
    each_acc=[]
    for i in range(num_folds):
        #除 i 之外的作为训练集，将第 i 份作为测试集
        pX_train=np.concatenate((X_train_folds[:i]+X_train_folds[i+1:]),axis=0)
        py_train=np.concatenate((y_train_folds[:i]+y_train_folds[i+1:]),axis=0)
        classifier.train(pX_train,py_train)
        dists=classifier.compute_distances_no_loops(X_train_folds[i])
        Yval_predict=classifier.predict_labels(dists,k=k)
        acc=np.mean(Yval_predict==y_train_folds[i])
        each_acc+=[acc]
    k_to_accuracies[k]=each_acc
```

SVM:

1. 完善使用 for 循环来评估多类 SVM 的损失函数 svm_loss_naive:

```
    dW /= num_train
    dW += 2* reg * W
```

2. 完善使用矢量化来评估多类 SVM 的损失函数 svm_loss_vectorized:

```
    num_train = X.shape[0]
    score = X.dot(W)
    #正确的分数 right_score
    right_score = score[np.arange(num_train),y].reshape(num_train,1)
    margins = np.maximum(0,score-right_score+1)
    #margin 中 label 值所在的误差归 0
    margins[np.arange(num_train),y] = 0
    #loss 取平均值，并加上正则化
    loss = np.sum(margins)
    loss /= num_train
    loss += reg * np.sum(W ** 2)

    #将 dS 初始化为 0，margins>0 的项（有误差的项）置为 1
    dS = np.zeros_like(score)
    dS[np.where(score-right_score+1>0)]=1
    dS[np.arange(num_train),y]= -1*(np.sum(score-right_score+1>0,axis=1)-1)
    #平均权重，正则化
    dW = X.T.dot(dS)
    dW /= num_train
    dW += 2*reg*W
```

3. 在 linear_classifier 中实现 SGD:

```
    def train(self, X, y, learning_rate=1e-3, reg=1e-5, num_iters=100,
                batch_size=200, verbose=False):
        num_train, dim = X.shape
        num_classes = np.max(y) + 1 # assume y takes values 0...K-1 where K is number of classes
        if self.W is None:
```

```
            self.W = 0.001 * np.random.randn(dim, num_classes)
        loss_history = []
        for it in range(num_iters):
            X_batch = None
            y_batch = None
            tp=np.random.choice(range(num_train),batch_size,replace=True)
            X_batch=X[tp,:]
            y_batch=y[tp]

            loss, grad = self.loss(X_batch, y_batch, reg)
            loss_history.append(loss)
            self.W = self.W - learning_rate * grad
            if verbose and it % 100 == 0:
                print('iteration %d / %d: loss %f' % (it, num_iters, loss))
        return loss_history

    def predict(self, X):
        y_pred = np.zeros(X.shape[0])
        scores= X.dot(self.W)
        y_pred= np.argmax(scores,axis=1)
        return y_pred
```

4. 完善 LinearSVM 利用交叉验证集调整超参数:

```
SVMmodel=LinearSVM()
for learning_rate in learning_rates:
    for reg in regularization_strengths:
        loss_hist=SVMmodel.train(X_train,y_train,
                                 learning_rate=learning_rate,
                                 reg=reg,num_iters=3000,
                                 verbose=False)
        y_train_pred = SVMmodel.predict(X_train)
        train_accuracy = np.mean(y_train == y_train_pred)
        y_val_pred = SVMmodel.predict(X_val)
        val_accuracy =np.mean(y_val == y_val_pred)
        if val_accuracy > best_val:
            best_val = val_accuracy
            best_svm = SVMmodel
        results[(learning_rate,reg)]=(train_accuracy,val_accuracy)
```

Softmax：

1. 完善使用 for 循环来评估 softmax 的损失函数 softmax_loss_naive:

```
def softmax_loss_naive(W, X, y, reg):
    loss = 0.0
    dW = np.zeros_like(W)
```

```
    num_train=X.shape[0]
    num_classes=W.shape[1]
    f=X.dot(W)
    f_max=np.reshape(np.max(f,axis=1),(num_train,1))
    prob=np.exp(f-f_max) / np.sum(np.exp(f-f_max),axis=1,keepdims=True)

    for i in xrange(num_train):
        for j in xrange(num_classes):
            if(j==y[i]):
                loss+=-np.log(prob[i,j])
                dW[:,j]+=(1-prob[i,j])*X[i]
            else:
                dW[:,j]-=prob[i,j]*X[i]
    loss /= num_train
    loss += 0.5 * reg * np.sum(W*W)
    dW =   dW/num_train + reg * W
    return loss, dW
```

2. 完善使用向量化来评估 softmax 的损失函数 softmax_loss_vectorized:

```
def softmax_loss_vectorized(W, X, y, reg):
    loss = 0.0
    dW = np.zeros_like(W)

    num_train=X.shape[0]
    num_classes=W.shape[1]
    f= X.dot(W)
    f_max=np.reshape(np.max(f,axis=1),(num_train,1))
    prob=np.exp(f-f_max) / np.sum(np.exp(f-f_max),axis=1,keepdims=True)
    loss = -np.sum(np.log(prob[range(num_train),list(y)]))
    loss /= num_train
    loss += 0.5 * reg * np.sum(W*W)

    dS= prob
    dS[range(num_train),list(y)] -=1
    dX=(X.T).dot(dS)
    dW = dW/num_train + reg * W

    return loss, dW
```

3. 利用交叉验证集调整超参数:

```
for learning_rate in learning_rates:
    for regularization_strength in regularization_strengths:
```

```
        softmax = Softmax()
        loss = softmax.train(X_train,y_train,
                             learning_rate=learning_rate,
                             reg=regularization_strength,
                             num_iters=1000,verbose=True)
        y_train_pred=softmax.predict(X_train)
        train_accuracy = np.mean(y_train==y_train_pred)
        y_val_pred=softmax.predict(X_val)
        val_accuracy = np.mean(y_val == y_val_pred)
        results[(learning_rate,regularization_strength)]=(train_accuracy,val_accuracy)
        if val_accuracy > best_val:
            best_val = val_accuracy
            best_softmax = softmax
pass
```

Three-layer-net:

1. 完善 loss 函数:

```
    def loss(self, X, y=None, reg=0.0):
        #.....已略去
        #forward pass
        Z1 = X.dot(W1) + b1
        A1 = np.maximum(0, Z1)
        Z2 = A1.dot(W2) + b2
        A2 = np.maximum(0, Z2)
        scores =   A2.dot(W3) + b3

        #.....已略去
        #forward pass Use the Softmax classifier loss
        scores -= np.max(scores, axis=1,keepdims=True)
        exp_scores = np.exp(scores)
        probs = exp_scores / np.sum(exp_scores,axis=1,keepdims=True)
        y_label = np.zeros((N,probs.shape[1]))
        y_label[np.arange(N),y] = 1
        loss = (-1) * np.sum(np.multiply(np.log(probs),y_label)) / N
        loss +=   reg * (np.sum(W1*W1) + np.sum(W2*W2) + np.sum(W3*W3))
        #.....已略去
        #Compute the backward pass
        dZ3 = probs-y_label
        dW3 = A2.T.dot(dZ3) / N + 2 * reg * W3
        db3 = np.sum(dZ3,axis=0) / N

        dZ2 = (dZ3).dot(W3.T) * (A2 > 0)
        dW2 = A1.T.dot(dZ2) / N + 2 * reg * W2
        db2 = np.sum(dZ2,axis=0) / N
```

```
        dZ1 = (dZ2).dot(W2.T) * (A1 > 0)
        dW1 = X.T.dot(dZ1) / N + 2 * reg * W1
        db1 = np.sum(dZ1,axis=0) / N
        grads['W3'] = dW3
        grads['b3'] = db3
        grads['W2'] = dW2
        grads['b2'] = db2
        grads['W1'] = dW1
        grads['b1'] = db1

        return loss, grads
```

2. 完善 train 函数:

```
batch_inx = np.random.choice(num_train,batch_size)
            X_batch = X[batch_inx,:]
            y_batch = y[batch_inx]
#.....
            self.params['W1'] -=   learning_rate * grads['W1']
            self.params['b1'] -=   learning_rate * grads['b1']
            self.params['W2'] -=   learning_rate * grads['W2']
            self.params['b2'] -=   learning_rate * grads['b2']
            self.params['W3'] -=   learning_rate * grads['W3']
            self.params['b3'] -=   learning_rate * grads['b3']
```

3. 完善 predict 函数:

```
score = self.loss(X)
      y_pred = np.argmax(score,axis=1)
      pass
```

4. 利用交叉验证集调整超参数

```
input_size = 32 * 32 * 3
num_classes = 10
learning_rates = [5e-3,1e-3,1e-4]
regs = [0.01,0.005,0.001]
hidden_sizes = [50,100,150]
num_iters = 10000
learning_rate_decay=0.99
batch_size=128
best_acc = -1
best_hyper_param = []

# Train the network
for hidden_size in hidden_sizes:
    net = ThreeLayerNet(input_size, hidden_size, num_classes)
    #for num_iters in num_iterss:
```

```
    for learning_rate in learning_rates:
        for reg in regs:
            stats = net.train(X_train, y_train, X_val, y_val,
                    num_iters=num_iters, batch_size=batch_size,
                    learning_rate=learning_rate, learning_rate_decay=learning_rate_decay,
                    reg=reg, verbose=False)
            val_acc = (net.predict(X_val) == y_val).mean()
            if val_acc>best_acc:
                best_acc = val_acc
                best_net = net
                best_hyper_param = [hidden_size,num_iters,learning_rate,reg]
                print('Temp best validation accuracy:',val_acc,'\t',
                      'best hyper param: ',[hidden_size,num_iters,learning_rate,reg])
print('Validation accuracy:',best_acc)
print('Best hyper parm:',best_hyper_param)
```

结论分析与体会：

完全矢量化比 for 循环运行速度更快，效率更高。
SVM 和 Softmax 的 score 一样，都是$S = XW$
SVM 损失函数：

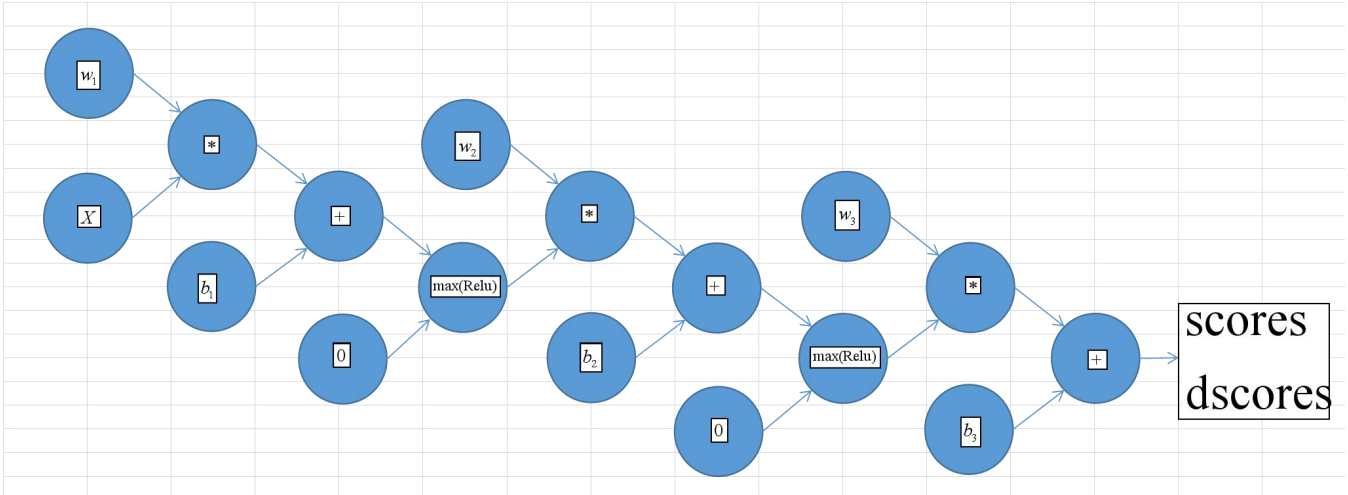
$$L = \frac{1}{N}\sum_i L_i + \lambda\sum_k\sum_l W_{k,l}^2 = \frac{1}{N}\sum_i\sum_{j\neq y_i}[\max(0, S_{i,j} - S_{i,y_i} + \Delta)] + \lambda\sum_k\sum_l W_{k,l}^2$$

Softmax 损失函数：

$$L = \frac{1}{N}\sum_i L_i + \lambda\sum_k\sum_l W_{k,l}^2 = \frac{1}{N}\sum_i -\log\left(\frac{e^{S_{y_i}}}{\sum_j e^{S_j}}\right) + \lambda\sum_k\sum_l W_{k,l}^2 = \frac{1}{N}\sum_i\left[-S_{y_i} + \log\left(\sum_j e^{S_j}\right)\right] + \lambda\sum_k\sum_l W_{k,l}^2$$

根据链式法则求导即可，记得在$\frac{\partial L}{\partial S}$中分 $j = y_i$ 和 $j \neq y_i$，另外记得加上正则项。

三层神经网络计算图：



就实验过程中遇到和出现的问题，你是如何解决和处理的，自拟 1－3 道问答题：

layer-net 反向传播计算梯度:

$$
\begin{cases}
dz_3 = dscores \\
dw_3 = A_2^T \cdot dz_3 + 2 \cdot reg \cdot w_3 \\
db_3 = dz_3 \\
dz_2 = dz_3 \cdot w_3^T (A_2 > 0) \\
dw_2 = A_1^T \cdot dz_3 + 2 \cdot reg \cdot w_2 \\
db_2 = dz_2 \\
dz_1 = dz_2 \cdot w_2^T (A_1 > 0) \\
dw_1 = X^T \cdot dz_2 + 2 \cdot reg \cdot w_1 \\
db_1 = dz_1
\end{cases}
$$