DaSE CV Assignment1 Part1

助教：刘鑫

数据科学与工程学院

2022.3.10

Outline

加载 Cifar-10 数据集

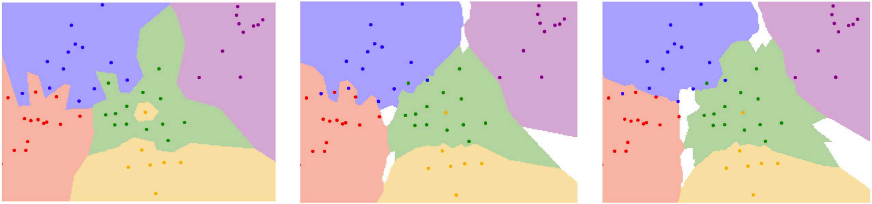
```
X_train, y_train, X_test, y_test = load_CIFAR10(cifar10_dir)
print('Training data shape: ', X_train.shape)
print('Training labels shape: ', y_train.shape)
print('Test data shape: ', X_test.shape)
print('Test labels shape: ', y_test.shape)
```

查看数据维度

```
Training data shape: (50000, 32, 32, 3)
Training labels shape: (50000,)
Test data shape: (10000, 32, 32, 3)
Test labels shape: (10000,)
```

K-Nearest Neighbors

Instead of copying label from nearest neighbor,
take **majority vote** from K closest points



K = 1 K = 3 K = 5

Figure: KNN Demo

KNN 算法实现

两层嵌套循环，遍历测试点，求其到全部训练样本点的距离

- 输入 X，(num_test,D) 维的数组
- 返回 dists (num_test, num_train) 维的 numpy 数组, dists[i, j] 表示测试样本 i 到训练样本 j 的距离

```
def compute_distances_two_loops(self, X):
      num_test = X.shape[0]
      num_train = self.X_train.shape[0]
      dists = np.zeros((num_test, num_train))
      for i in range(num_test):
         for j in range(num_train):
            difference = X[i]-self.X_train[j] # (x1-y1,x2-y2,x3-y3...xn-yn)
            difference_square = difference ** 2 # ((x1-y1)^2,(x2-y2)^2...(xn-yn)^2)
            distance = np.sqrt(np.sum(difference_square))
                  #sqrt((x1-y1)^2+(x2-y2)^2...+(xn-yn)^2)
            dists[i,j] = distance
            # *****END OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****
      return dists
```

KNN 算法实现

单层循环，遍历测试点，求其到全部训练样本点的距离

- 输入输出同上

```
def compute_distances_one_loop(self, X):
    num_test = X.shape[0]
    num_train = self.X_train.shape[0]
    dists = np.zeros((num_test, num_train))
    for i in range(num_test):
        #######################################################################
        # TODO:                                                               #
        # Compute the l2 distance between the ith test point and all training #
        # points, and store the result in dists[i, :].                        #
        # Do not use np.linalg.norm().                                        #
        #######################################################################
        # *****START OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****

        difference = self.X_train - X[i] #broadcast
        distance = np.sum(difference ** 2,axis = 1) #按行求和
        dists[i,:] = np.sqrt(distance)

        # *****END OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****
    return dists
```

KNN 算法实现

完全向量化运行，求其到全部训练样本点的距离

- 输入输出同上

```
def compute_distances_no_loops(self, X):
    num_test = X.shape[0]
    num_train = self.X_train.shape[0]
    dists = np.zeros((num_test, num_train))
    #########################################################################
    # HINT: Try to formulate the l2 distance using matrix multiplication #
    #     and two broadcast sums.                                       #
    #########################################################################
    # *****START OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****
    train_square = np.sum(self.X_train ** 2,axis = 1,keepdims=True)
    train_square = np.broadcast_to(train_square,shape=(num_train,num_test)).T
    test_square = np.sum(X**2,axis=1,keepdims=True)
    test_square = np.broadcast_to(test_square, shape=(num_test,num_train))
    cross = np.dot(X,self.X_train.T) #交叉项
    dists = np.sqrt(train_square + test_square - 2*cross) #sqrt
    # *****END OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****
    return dists
```

Cross Validation

交叉验证

将数据集划分 3 个部分，两两之间没有交集

- 训练集 Training Set
- 验证集 Validation Set
- 测试集 Test Set

fold 1	fold 2	fold 3	fold 4	fold 5	test
fold 1	fold 2	fold 3	fold 4	fold 5	test
fold 1	fold 2	fold 3	fold 4	fold 5	test

Figure: K-Fold Cross Validation

Cross Validation 实现

实现交叉验证，帮助选取超参数

```
num_folds = 5
k_choices = [1, 3, 5, 8, 10, 12, 15, 20, 50, 100]
X_train_folds = []
y_train_folds = []
###########################################################
# TODO:                                                    #
# Split up the training data into folds. After splitting, X_train_folds and #
# y_train_folds should each be lists of length num_folds, where #
# y_train_folds[i] is the label vector for the points in X_train_folds[i]. #
# Hint: Look up the numpy array_split function.            #
###########################################################
# pass
y_train_ = y_train.reshape(-1, 1)
X_train_folds , y_train_folds = np.array_split(X_train, 5), np.array_split(y_train_, 5)
############################################################
#                         END OF YOUR CODE                 #
############################################################
```

Cross Validation 实现

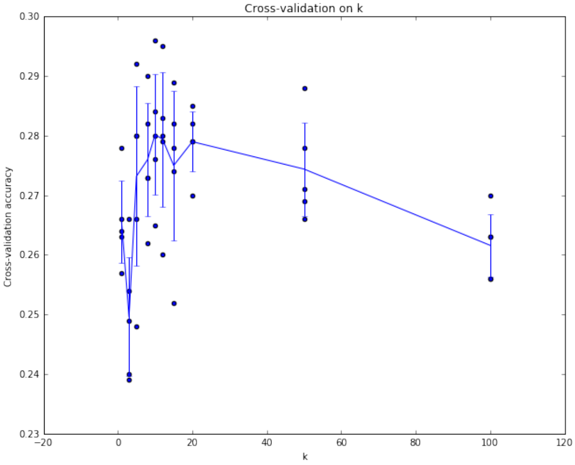
实现交叉验证，帮助选取超参数

```
k_to_accuracies = {} # Store the accuracies for all fold and all values of k
#TODO#
# Perform k-fold cross validation to find the best value of k. For each
# possible value of k, run the k-nearest-neighbor algorithm num_folds times,
# where in each case you use all but one of the folds as training data and the
# last fold as a validation set.
for k_ in k_choices:
   k_to_accuracies.setdefault(k_, [])
for i in range(num_folds):
   classifier = KNearestNeighbor()
   X_val_train = np.vstack(X_train_folds[0:i] + X_train_folds[i+1:])
   y_val_train = np.vstack(y_train_folds[0:i] + y_train_folds[i+1:])
   y_val_train = y_val_train[:,0]
   classifier.train(X_val_train, y_val_train)
   for k_ in k_choices:
      y_val_pred = classifier.predict(X_train_folds[i], k=k_)
      num_correct = np.sum(y_val_pred == y_train_folds[i][:,0])
      accuracy = float(num_correct) / len(y_val_pred)
      k_to_accuracies[k_] = k_to_accuracies[k_] + [accuracy]
#                         END OF YOUR CODE                         #
```

Cross Validation Cont'd

Error bar

基于可视化结果，直观地选取超参



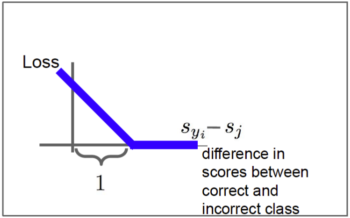
Multiclass Support Vector Machine

Suppose: 3 training examples, 3 classes.
With some W the scores $f(x, W) = Wx$ are:



cat	**3.2**	1.3	2.2
car	5.1	**4.9**	2.5
frog	-1.7	2.0	**-3.1**

Interpreting Multiclass SVM loss:



$$L_i = \sum_{j \neq y_i} \begin{cases} 0 & \text{if } s_{y_i} \geq s_j + 1 \\ s_j - s_{y_i} + 1 & \text{otherwise} \end{cases}$$

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

Figure: Hinge Loss

Multiclass Support Vector Machine

Multiclass SVM Loss

$$L_i = \sum_{j \neq y_i} \max\left(0, S_j - S_{y_i} + \Delta\right) \tag{1}$$

- L_i 第 i 个样本的 Hinge Loss
- S_{y_i} 第 i 个样本分类标签分数
- S_j 第 i 个样本对应的标签

Regularized Multiclass SVM Loss Cont'd

$$L_i = \sum_{j \neq y_i} \max\left(0, S_j - S_{y_i} + \Delta\right) + \frac{\lambda}{2}\|w\|^2 \tag{2}$$

SVM 算法实现

Naive SVM Loss

```
def svm_loss_naive(W, X, y, reg):
  dW = np.zeros(W.shape) # initialize the gradient as zero
  # compute the loss and the gradient
  num_classes = W.shape[1]
  num_train = X.shape[0]
  loss = 0.0
  for i in xrange(num_train):
    scores = X[i].dot(W)
    correct_class_score = scores[y[i]]
    for j in xrange(num_classes):
      if j == y[i]:
        continue
      margin = scores[j] - correct_class_score + 1 # note delta = 1
      if margin > 0:
        loss += margin
        dW[:,j] += X[i].T
        dW[:,y[i]] += -X[i].T
  # Right now the loss is a sum over all training examples, but we want it
  # to be an average instead so we divide by num_train.
  loss /= num_train
  dW /= num_train
  # Add regularization to the loss.
  loss += 0.5 * reg * np.sum(W * W)
  dW += reg * W
  return loss, dW
```

SVM 算法实现

Vectorized SVM Loss

```
num_train = X.shape[0] # 得到样本的数目
scores = np.dot(X, W) # 计算所有的得分
y_score = scores[np.arange(num_train), y].reshape((-1, 1)) # 得到每个样本对应label的得分
mask = (scores - y_score + 1) > 0 # >0 loss 下标
scores = (scores - y_score + 1) * mask
loss = (np.sum(scores) - num_train * 1) / num_train # 去掉每个样本多加的对应label得分，然后平均
loss += reg * np.sum(W * W)


# dw = x.T * dl/ds
ds = np.ones_like(scores) # 初始化ds
ds *= mask # >0 ->1, <0 ->0
ds[np.arange(num_train), y] = -1 * (np.sum(mask, axis=1) - 1) #
# 负非0项个数
dW = np.dot(X.T, ds) / num_train
dW += 2 * reg * W # add regularization
```

Softmax

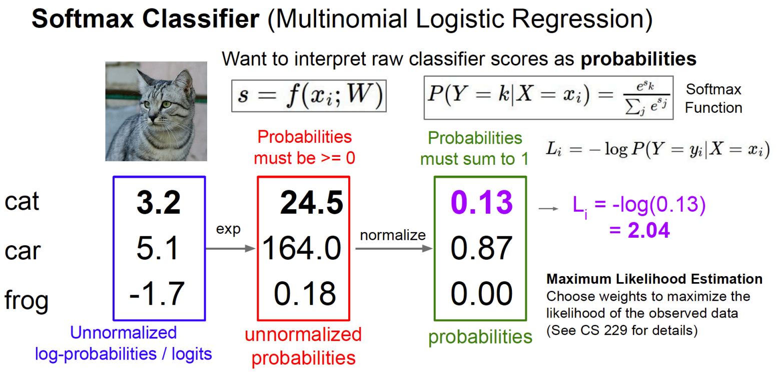


Figure: Softmax

Softmax Cont'd

Softmax 的各分量

$$p_k = \frac{e^{f_k}}{\sum_j e^{f_j}} \tag{3}$$

- k : 某个特定类别
- f : 分值向量
- j : 任意一个类别
- p_k : k 类别的概率大小

Regularized Softmax Loss

$$L = \underbrace{\frac{1}{N}\sum_i L_i}_{\text{data loss}} + \underbrace{\frac{1}{2}\lambda \sum_k \sum_l W_{k,l}^2}_{\text{regularization loss}} \tag{4}$$

Softmax 算法实现

Naive Softmax

```python
def softmax_loss_naive(W, X, y, reg):
loss = 0.0
dW = np.zeros_like(W)
for i in range(X.shape[0]):
   score = np.dot(X[i], W)
   score -= max(score) # stability
   score = np.exp(score) # exp
   softmax_sum = np.sum(score) # get den
   score /= softmax_sum
   # calc gradient
   for j in range(W.shape[1]):
      if j != y[i]:
         dW[:, j] += score[j] * X[i]
      else:
         dW[:, j] -= (1 - score[j]) * X[i]

   loss -= np.log(score[y[i]]) # cross entropy
loss /= X.shape[0]
dW /= X.shape[0]
loss += reg * np.sum(W * W) # add reg
dW += 2 * reg * W
return loss, dW
```

Softmax Cont'd

Softmax 算法实现

Vectorized Softmax

```
def softmax_loss_vectorized(W, X, y, reg):
"""
Softmax loss function, vectorized version.
Inputs and outputs are the same as softmax_loss_naive.
"""
# Initialize the loss and gradient to zero.
loss = 0.0
dW = np.zeros_like(W)

scores = np.dot(X, W) # calc score
scores -= np.max(scores, axis=1, keepdims=True) # stability
scores = np.exp(scores) # exp
scores /= np.sum(scores, axis=1, keepdims=True) # softmax
ds = np.copy(scores) # init grad
ds[np.arange(X.shape[0]), y] -= 1 # get grad
dW = np.dot(X.T, ds) # get grad for w
loss = scores[np.arange(X.shape[0]), y] # compute loss
loss = -np.log(loss).sum() #get cross entropy
loss /= X.shape[0]
dW /= X.shape[0]
loss += reg * np.sum(W * W)
dW += 2 * reg * W
return loss, dW
```

问答环节

Questions!