Weekly Report(Apr 23-Apr 29)

Liu Junnan

Abstract

This week I started to learn computer vision on CS231n, and assignment 1 of the course.

1 Work Done

1.1 Computer Vision

This week I started CS231n course addressed by Fei Fei Li. Computer vision is an interesting and challenging field of computer science. The most information we get everyday is visual information, and we are so easy to recognize and analyze these information that we can't realize how hard it is for computers to understand visual data, since what computers actually get are just 3 dimensional matrices. What computer vision algorithms try to do is to apply machine learning algorithms to solve these problems.

1.2 K Nearest Neighbor Algorithm

CS231n again walks through the pipeline of machine learning task. The first assignment requires us to implement K-nearest neighbors algorithm. The principle of KNN is easy to understand. The training process is to memorize all the training data, and the predicting process is to compute all the distances between training set and the test data, and find the smallest k data, which are so-called k nearest neighbors. Although as simple as it is, there are still many tricks in implementation.

1.3 Implementation of KNN algorithm

The assignment asks us to implement KNN prediction function ,that is, computing distances among the test data and all the training data, in three different versions – two loops, one loop, and no loop. The test data have shape (num_test, D), where num_test is the number of the test points, and D is the number of features of test and also training data. The training data have shape (num_train, D). Therefore the distance matrix dist has shape (num_test, num_train).

The two loop version of implementation is quite straightforward. It just iterates all the test data and training data.

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

The one loop version just iterates test data, and computes the distances between this data and all the training data, with the help of broadcast feature of numpy.

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

These two versions are not so difficult, but the no loop version demands full vectorized computation, which makes it much trickier to implement. To be honest I didn't figure it out, so I had to search the Internet and then found the solution, which makes good use of broadcast feature of numpy. The code is as follows:

```
P = np.sum(X**2, axis=1)
Q = np.sum(self.X_train**2, axis=1)
PQ = X.dot(self.X_train.T)
dists = np.sqrt(np.transpose([P]) + Q - 2*PQ)
```

Let p denotes test data matrix, q training data matrix, and $dist$ distance matrix. The equation of distance is $(p-q)^2$. There is no way to just simple subtract test and training data and square the subtraction because they don't have the same shape. However, the solution applies the equation $(p \pm q)^2 = p^2 \pm 2pq + q^2$, and the term p^2 and q^2 are easy to vectorize, the middle term $2pq$ is dot multiplication of two matrix, so it is also vectorized. But actually these three terms have totally different shapes. In code line 1 above, P has shape (num_test,) — which in numpy indicates that this array has 1 dimension with size num_test. Similarly, Q has shape (num_train,), and PQ has shape (num_test, num_train). Here is the trick. if we compute $[P].T - Q$ instead, these two array will be broadcast, and yields result with shape (num_test, num_train).

To further explain what happens, it is important to read and understand the rules of broadcasting.

- When operating on two arrays, NumPy compares their shapes element-wise. It starts with the trailing dimensions, and works its way forward. Two dimensions are compatible when
  1. they are equal, or
  2. one of them is 1
- Arrays do not need to have the same number of dimensions. When either of the dimensions compared is one, the other is used. In other words, dimensions with size 1 are stretched or copied to match the other.

Let check the rules. P has shape (num_test,), and we add brackets $[P]$ makes it a 2 dimensional array with shape (1, num_test), and function np.transpose() transposes it yielding shape (num_test, 1). Since Q has shape (num_train,), it matches the last dimension of the term np.transpose([P]), so the rule 1 of broadcasting is met; np.transpose([P]) has more dimensions than Q, so the rule 2 is met. Therefore the result finally has shape (num_test, num_train), which is the same as PQ, so they will do element-wise subtraction.

The assignment then compare the time cost among these three versions of implementation. The result shows marvelous improvement that vectorization does. This experiment again demonstrates

two loops	one loop	no loops
36.73	121.6	0.33

Table 1: Performance Compare

the power of vectorization in matrix computation. We should definitely vectorize the code whenever possible.

2 Plans

In the next week I will finish assignment 1 of CS231n, and continue the course.