Assignment1 Report

August 25, 2020

1 Euclidean distance function

Listing 1 is the code snippet of out euclidean distance function. It works by expanding following expression:

$$||\vec{x_a} - \vec{z_b}||^2 = ||\vec{x_a}||^2 + ||\vec{z_b}||^2 - 2(\vec{x_a} \cdot \vec{z_b})$$

Where $\vec{x_a}, \vec{z_b}$ are a^{th} and b^{th} row vectors in matrix $\boldsymbol{X}$ and $\boldsymbol{Z}$, respectively. The result matrix $\boldsymbol{D}$ where $\boldsymbol{D}_{a,b} = ||\vec{x_a} - \vec{z_b}||^2$ can then be calculated as sum of three matrices:

$$\boldsymbol{D} = \begin{bmatrix} ||\vec{x_1}||^2 \\ ||\vec{x_2}||^2 \\ \vdots \\ ||\vec{x_{N_1}}||^2 \end{bmatrix} \times \begin{bmatrix} 1 & 1 & \dots & 1 \end{bmatrix} + \begin{bmatrix} ||\vec{z_1}||^2 & ||\vec{z_2}||^2 & \dots & ||\vec{z_{N_2}}||^2 \end{bmatrix} \times \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix} - 2\boldsymbol{X}\boldsymbol{Z}^T$$

In implementation, we use TensorFlow broadcast mechanism to expand two norm vectors into matrices without actually constructing two vectors full of 1.

2 Making Predictions for Regression

2.1 Choosing the nearest neighbors

Listing 2 is the responsibility function we implemented. To make full use of the vectorized TensorFlow method $tf.nn.top\_k()$, a negation must be applied to input distance matrix m to obtain a list of k elements with the least amount of distance to the point of interest, since $top\_k()$ attempts to find the largest values in the input list. The graph must then be run to compute the value and export to a numpy array for post processing.

The code in line 22 generates a vector with $N_1 \times k$ elements where i^{th} element is $\lfloor \frac{i-1}{k} \rfloor$ so that integer array indexing can be used to assign all responsibility value to $\frac{1}{k}$ without a loop, and the rest of matrix will be zero.

```python
def a1_p1_distance(x,z):
  """Get Euclidean distance of two group of vectors

  input:
    x   Matrix of dimension N_1*D in form of [x(1)^T, x(2)^T, ...,x(N_1)^T]
      Where x(i)^T is a row vector of dimension D
    z   Matrix of dimension N_2*D in form of [z(1)^T, z(2)^T, ...,z(N_2)^T]
      Where z(i)^T is a row vector of dimension D

  return a matrix d with dimension N_1*N_2 where d_{i,j} is the squared
  euclidean distance between x(i) and z(j)

  return values are TensorFlow tensors while input matrices can be either
  Numpy arrays or tensors
  """

  #x_sqr_norm is an N_1*1 matrix
  #z_sqr_norm is a vector with N_2 elements
  x_sqr_norm=tf.square(tf.norm(x,axis=1,keep_dims=True))
  z_sqr_norm=tf.square(tf.norm(z,axis=1,keep_dims=False))

  # just for debug purpose (verify the dimensions)
  # print(x_sqr_norm.get_shape())
  # print(z_sqr_norm.get_shape())

  return (x_sqr_norm+z_sqr_norm)+(-2)*tf.matmul(x,z,transpose_b=True)
```

Listing 1: Vectorized euclidean distance function

```python
def a1_p2_q1_responsibility(m,k,s):
  """Take a pairwise distance matrix and return responsibilities

  input:
    m   A pairwise distance matrix of dimension N_1*N_2, probably from
      a1_p1_distance(). This should be a TensorFlow tensor.
    k   An integer scalar as parameter k
    s   A TensorFlow session for evaluating m
  return:
    A matrix of dimension N_1*N_2, where in each row:
      The k smallest values are 1/k
      Other (N_2-k) values are set to 0
    The return value will be a Numpy array
  """
  m_negated=(-1)*m #so we find max k items in negated to find min k in src
  m_dims=m.get_shape().as_list()
  _, indices_tensor=tf.nn.top_k(m_negated,k)
  indices_nparray=s.run(indices_tensor)
  indices_col=indices_nparray.flatten()

  rowCount=m_dims[0]
  indices_row=np.arange(0,rowCount,1.0/k).astype(int)

  result=np.zeros(m_dims)
  result[indices_row,indices_col]=1.0/k
  return result
```

Listing 2: Vectorized responsibility function

Table 1: k-NN MSE losses for data1D dataset

k	Training	Validation	Test
1	0.0000	0.5580	0.4970
3	0.1734	0.3908	0.2871
5	0.2129	0.3340	0.2500
50	0.2625	0.2822	0.2739

Figure 1: k-NN predictions for various k

2.2 Prediction

Table 1 lists the losses for various k values. $k = 50$ gives the lowest validation error, and is the best choice.

Figure 1 is the plot of prediction function for different k values over the same x in training data. The blue dots are training data and orange dots are predictions. One can see that when $k = 1$, the prediction is identical to training data, but when k increases, the prediction becomes less susceptible to noises in training data and it grasps the smooth underlying function well. Hence in this case, the largest value $k = 50$ should be chosen.

3 Making Predictions for Classification

Due to the observation that name and gender classification uses the same set of images, we share the similarity data when doing k-NN search. Our code works at name and gender recognition at the same time and in parallel, and the following analysis will also be performed in parallel.

3.1 Predicting class label

We tried to reduce unnecessary computations by finding tensors that can be shared. Listing 3 is the main part of graph building code.Our first observation is that the negated pairwise distance matrix can be shared for all k values, so we build these two tensor outside the for loop of k (one for validation set, the other for test set). We also exploited the fact that the nearest k neighbors are the same for name and gender recognition.

We expanded the responsibility function directly into the caller (since the amount of code left unchanged becomes limited), and in this way we managed to express all our logic in TensorFlow graph, without bothering evaluating a single tensor in the middle. We also build TensorFlow graph to get name for validation, name for test, gender for validation, and gender for test sets in parallel. After the graph is built, we only need to evaluate the 4 prediction tensors at the end of graph building code.

```
  k_list=[1,5,10,25,50,100,200]
  # all results (to be stacked)
  list_name_pred_vl=[]
  list_name_pred_ts=[]
  list_gender_pred_vl=[]
  list_gender_pred_ts=[]

  for k in k_list:
    _, indices_t_vl2tr=tf.nn.top_k(neg_dist_vl2tr,k)
    _, indices_t_ts2tr=tf.nn.top_k(neg_dist_ts2tr,k)
    indices_lin_vl2tr=tf.reshape(indices_t_vl2tr,[-1])
    indices_lin_ts2tr=tf.reshape(indices_t_ts2tr,[-1])

    name_pool_vl2tr=tf.reshape(tf.gather(tr_target_name,indices_lin_vl2tr),[-1,k])
    name_pool_ts2tr=tf.reshape(tf.gather(tr_target_name,indices_lin_ts2tr),[-1,k])
    gender_pool_vl2tr=tf.reshape(tf.gather(tr_target_gender,indices_lin_vl2tr),[-1,k])
    gender_pool_ts2tr=tf.reshape(tf.gather(tr_target_gender,indices_lin_ts2tr),[-1,k])

    list_name_pred_vl.append(majority_vote_in_each_row(name_pool_vl2tr))
    list_name_pred_ts.append(majority_vote_in_each_row(name_pool_ts2tr))
    list_gender_pred_vl.append(majority_vote_in_each_row(gender_pool_vl2tr))
    list_gender_pred_ts.append(majority_vote_in_each_row(gender_pool_ts2tr))

  t_name_pred_vl=tf.stack(list_name_pred_vl)
  t_name_pred_ts=tf.stack(list_name_pred_ts)
  t_gender_pred_vl=tf.stack(list_gender_pred_vl)
  t_gender_pred_ts=tf.stack(list_gender_pred_ts)
```

Listing 3: k-NN face recognition graph building code snippet

```
def majority_vote_in_each_row(t):
  l=[]
  t_unstacked=tf.unstack(t)
  for row in t_unstacked:
    val,_,count=tf.unique_with_counts(row)
    result=tf.reshape(tf.gather(val,tf.argmax(count)),[])
    l.append(result)
  return tf.stack(l)
```

Listing 4: helper function majority_vote_in_each_row()

Table 2: Face recognition: Number of correct recognitions in each set

k	Name		Gender	
	validation set (out of 92)	test set (out of 93)	validation set (out of 92)	test set (out of 93)
1	61	66	84	86
5	56	64	84	84
10	53	62	82	83
25	55	61	83	82
50	53	54	82	80
100	44	46	79	80
200	29	37	72	72

Figure 2: K-NN face recognition failure cases in test sets when $k = 10$

(a) #1, name(0) and gender(1) incorrect

(b) #2, name(4) incorrect

(c) #5, name(1) incorrect

(d) #11, name(4) and gender(0) incorrect

3.2 Face and Gender recognition using k-NN

Table 2 are face and gender recognition results for listed k values. $k = 1$ gives best results for both face and gender recognition in both test and validation sets. $k = 1$ for face recognition gives 70.968% accuracy (66 correct) in 93 test points, and in gender recognition it gives 92.473% accuracy (86 correct) in 93 test points.

3.3 Face and Gender recognition failure case study

Figure 2 has 4 recognition failure cases. The images at the top row are the images in test set, and each 10 images below are the 10 "closest" images. It is not hard to see that the similarity of image in these cases do not corresponds to "similarity" of face in image, and usually only first one or two corresponds to the face in test image well. This indicates that k-NN is not suitable for face recognition, and if we have to use k-NN for face recognition then the smallest k value should be used.