Assignment 3

Author : Zhou Fang

studentId : 6286914

Part 1 : Genetic Programming

1.Introduction

Genetic algorithm is a very popular machine learning in recent year, which based on the idea of evolution in the nature. And it simulates the evlolution of the population in the natural world. In the computer, it Simulate a population by initializing a certain number datastruct, and by computing the fitness of each individual to get the adaptability to the environment, and select the better adaptability individual to do the evolution work which by cross over their 'DNA' sequence, In the end of the evolution, each individual has certain probability to mutation. By this process mentioned above, computer can simulatethe population evolution. In this assignment we need to solve the Traveling Sale Person Problem by using the genetic algorithm, which can find path between the all given city which has the lowest cost. the dataset gives us a amount number of city, and each city has a certain type, different type of city has a differet cost factor, the real cost will be the cost factor times the distance between two city. Therefore, base on the above, we need to find a best path y using the genetic programming

2. implement

In my experiment , i implement the following step to achieve the goal.

before we do the following step, I use a class called the 'City' to stored the data of the dataset file, which will have the benefit that the reading the data and storing the results will be very clear, and using this data structure in future functions **will be more clear and simpler than array-based**.

```
// def class of city
class City{
    public :
        int x_pos;
        int y_pos;
        int type;
    public :
    // construstor
    City(){}
    // construstor
```

```
    City(int x_pos, int y_pos, int type):x_pos(x_pos), y_pos(y_pos),
type(type){}
    // output reload
    friend ostream& operator<<(ostream& out,const City& c){
        out << "x_pos: " << c.x_pos << " y_pos: " << c.y_pos << " type: "<<
c.type<<endl;
        return out;
    }
    // reload the equal operator
    bool operator==(const City& c){
        return (x_pos==c.x_pos) && (y_pos==c.y_pos) && (type==c.type);
    }

};
```

2.1 Step1 : some basic gentic algorithm

- Initialize the population

  In realizing population evolution, we must first have a population of a certain size. I Initialize the population with completely random path which can link all the city in a certain population size. and also initialize the next population with 0 **with population size which is read from the data file** which can make all the data container fit all the given data file which may have a different city number.

```
void init_pop(vector<vector<int> > &CurrentPop, vector<vector<int> >
&NextPop, vector<double> &Fitness, vector<int> &BestMember){
    srand(Seed);
    // init vector with 0
    NextPop.resize(cPopSize);
    Fitness.resize(cPopSize);
    BestMember.resize(cIndividualLength);

    vector<int> init_path;
    // init path
    for(int i=0; i<cIndividualLength; i++){init_path.push_back(i);}
    for(int i=0; i<cPopSize; i++){
        vector<int> tmp_path = init_path;

        // init to the random path
        for(int j=0; j<cIndividualLength; j++){
            int random_pos = RandInt(cIndividualLength);
            // swap
            int tmp = tmp_path[j];
            tmp_path[j] = tmp_path[random_pos];
            tmp_path[random_pos] = tmp;
        }
        CurrentPop.push_back(tmp_path);
    }
```

```
    return ;
}
```

- Crossover between the two individual

  In this section, there will be a little difference compared to the crossover part of traditional genetic programming. In the traditional way, they just simply swap a certain length of the 'DNA' sequence between the two pranent 'DNA' sequences to generate the child's 'DNA'. However this will meet a truble if we doing so in our experiment, because if we just randomly swap some sequence in the parents' 'DNA', there will be **large probability to contain the same city in the child's 'DNA' sequence which is unreasonable in this TSP problem**. Therefore we need to figure out a method to generate child's 'DNA' with all different city too. In my experiment, i achieve this by the method below:

  - first we randomly select a certain length in one of the parent called 'cp1'(**this choosen section is consist of discrete potions of the parent rather then the continues sequence''**)
  - then we put the 'cp1' into the front of the child's 'DNA' sequence to form the part one of the child's 'DNA'
  - then we **choose points from the parents two which should be not exist in the part one of the child' 'DNA'**, and put those points into the child's 'DNA' sequence after part one with the same order with the parent two

  For example :

```
parent1 : 1 5 4 2 3
parent2 : 3 2 5 1 4
// choose 2, 4 from parent 1 put into the front of the child
child : 2, 4
// then choose points form parent2 which are differnt from the points
exist, and put // it into the back of child
// choose 3, 5, 1 which not exist in the child and put
child : 2, 4, 3, 5, 1
```

  After implementing the above steps, we can successfully perform the crossover operation in this experiment. The code can is shown below:

```
void CrossOver(vector<int> parent1, vector<int> parent2, vector<int>
&child1, vector<int> &child2, int Gen){
    // cCrossOverLengthRate need to be smaller and smaller
    int CrossOverLength = CurtCrossOverLength(Gen);
    // random to cross over
    // select the points which need to be swapped
    vector<int> CrossChoose;
    for (int i=0; i<cIndividualLength; i++) CrossChoose.push_back(i);
    // random swap it
    for (int i=0; i<CrossChoose.size();i++){
        int ran_pos = RandInt(cIndividualLength);
```

```
            int tmp = CrossChoose[i];
            CrossChoose[i] = CrossChoose[ran_pos];
            CrossChoose[ran_pos]= tmp;
        }
        // the front CrossOverLength is the swap points' pos
        vector<int> CrossPoints;
        for(int i=0; i<CrossOverLength; i++)
    {CrossPoints.push_back(CrossChoose[i]);}

        // start cross over
        child1.clear();
        child2.clear();

        // Cross Over
        for(int i=0;i<CrossOverLength;i++){
            child1.push_back(parent1[CrossPoints[i]]);
            child2.push_back(parent2[CrossPoints[i]]);
        }
        for(int i=0;i<cIndividualLength;i++){
            if(!IsExistElem(child1, parent2[i])){
                child1.push_back(parent2[i]);
            }
            if(!IsExistElem(child2, parent1[i])){
                child2.push_back(parent1[i]);
            }
        }
        if (child1.size()!=cIndividualLength ||
    child2.size()!=cIndividualLength){
            cout<< "Something Wrong When Crossing Over";
            exit(1);
        }
        return ;
    }
```

- Mutate the individual

  In my understanding, the mutaion operation prevents a population from falling into a local minimum. whic can make next population not too concentrated, so that it can be easier to evolve in all directions which helps a lot to achieve the global minimum.

  In my experiment, i provided four type of mutation

```
enum MutationType{OnePoint, TwoPoint, ThreePoint, ReducePoint};  //
the type of mutation
```

  - OnePoint means we just randomly choose one point, and randomly swap with another point
  - TwoPoint means we randomly choose two different points and swap with two

another points
    - ThreePoint means we randomly choose three different points and swap with three another points
    - ReducePoint can make the Mutation point reduce with the iteration incresse, this method will be mentioned in the futher step

- Evaluate the Fitness

  In this TSP problem, we use the the number which mulitipy all the distances along the path by the cost between cities as the fitness of each individual. So compare to the traditional fitness method, there will be a little different. **because in our experiment, lower fitness means better results which is different from the traditional way**

  In our experment, we calculate the Fitness by the code below

```
// compute the fitness of one member
double EvaluateFitness(vector<int> Member){
    unsigned long length = Member.size();
    double fitness = 0;
    for(int i=0; i<length-1; i++){
        fitness += get_distacne(AllCity[Member[i]],
AllCity[Member[i+1]],                          Member[i],
Member[i+1]);
    }
    return fitness;
}
```

2.2 Step2 : Lookup table : make a calculate cache

In process of calculating each individual's fitness, We are likely to need to run a lot of basic operations which will may appear repeatedly and during the calculation of Fitness. For example, calculate the distance between city A with city B may appear many times in our program. Therefore we can buld a table for cache which can store the distance value which has calculated before, so that we can read the cache directly in the next time appears.

```
// look up table is defined as a map
// cache to store the comptued distance
// the pair <int, int > iIndicate the <City_a_index, City_b_index>
map<pair<int, int>, double> look_up;
```

then we store a cache when calculate every distance:

```
double get_distacne(City c1, City c2, int index1, int index2){

    // if has cache then return directly
    if (look_up.count(pair<int, int>(index1, index2))) return
look_up[pair<int, int>(index1, index2)];

    // if not has stroe into the cache
    double dis;
    dis = sqrt(pow(c1.x_pos - c2.x_pos, 2) + pow(c1.y_pos - c2.y_pos, 2));
    dis *= type_cost[pair<int, int>(c1.type, c2.type)];
    // the distance between <City_a, City_b> should be the same with
<City_b, City_a>
    look_up[pair<int, int>(index1, index2)] = look_up[pair<int, int>
(index2, index1)] = dis;
    return dis;
}
```

2.3 Step3 : Two selection pattern : tournament and roulettewheel

Before the Crossover part, we need to select two parent first. Obviously, we cannot just choose two individual in the population which will lost the purpose of the evolution to the population. Therefore we need to select the better individual in current population to generate the child which apply to the next population.

In my experiment, there two method to do the selection

```
enum SelectType{tournament, roulettewheel};  // the type of Selection
Winner
```

- Tournament

  In Tournament, we randomly select n which can be defined as the 'cTournamentSize' individual each time. and select the best individual as a winner of this round of tournment

  ```
  // get the index if the winner DNA
  int Tournament(vector<double> Fitness){
      double WinFit = __DBL_MAX__;
      int Winner;
      for(int i=0; i<cTournamentSize; i++){
          int random_member = RandInt(cPopSize);
          if(Fitness[random_member] < WinFit){
              WinFit = Fitness[random_member];
              Winner = random_member;
          }
      }
      return Winner;
  }
  ```

- RouletteWheel

  In roulettewheel, we just choose the winner by the probability of each individual exist in the current population, which means that **the individual with better fitness have higher probability to be choosen**

  There we do need to calculate the probability of each individual to be choosen. However there is a trouble in our experiment when calculating the probability, **because the lower fitness means lower cost which will have higher probability to be choose, which means that we can not use the traditional method to calculate the probability** . Therefore, when we do the RouletteWheel, we do follow steps

  - Normalised all the fitness of each individual
  - Calculate the total fitness after Normalition
  - Minus each by the max fitness which can give the lower fitness higher value
  - Divide by the total fitness then we get the probability

```
int RouletteWheel(vector<double> Fitness){
    // find the min and max number in Fitness
    double min_f = Fitness[0], max_f = Fitness[0];
    // index
    int max_i = 0;

    for(int i=1; i<cPopSize; i++){
        if(Fitness[i] > max_f){
            max_f = Fitness[i];
            max_i = i;
            continue;
        }
        if(Fitness[i] < min_f){
            min_f = Fitness[i];
            min_f = i;
        }
    }
    // normalised
    vector<double> Norm_Fitness(cPopSize);
    for(int i=0; i<cPopSize; i++){
        Norm_Fitness[i] = Fitness[i] - min_f;
    }

    // Reverse Norm_Fitness
    // This is important cause in this experments The Lower Fitness
mean the better answer
    // Thus we need to put the lower Fitness to have higher possbility
to be choosen
    double norm_max = Norm_Fitness[max_i];
    // liner substart
    double Tot = 0;
    for(int i=0; i<cPopSize; i++){
        // make lower fitness high possbility
```

```
            Norm_Fitness[i] = norm_max - Norm_Fitness[i];
            Tot += Norm_Fitness[i];
        }
        // make it to possbility [0,1]
        for(int i=0; i<cPopSize; i++){Norm_Fitness[i] /= Tot;}

        // now we get the Roulette Wheel
        double possibility = Rand01();
        double m = 0;
        for(int i=0; i<cPopSize; i++){
            m += Norm_Fitness[i];
            if (possibility < m)
                // get winner
            {
//                cout << i << endl;
                return i;
            }
        }
        cout << "Something Wrong in the Roulette Wheel";
        exit(1);
    }
```

2.4 Step 4 : Make the crossover rate and Mutation potions reduce as evolution progresses

As the evolutionary steps increase, we need crossover and mutaion to have less and less impact on our children, because it makes it easier to preserve the dominant genes.

- In the Crossover part :

```
// we add a paramter to controll if process this character
const bool IS_REDUCE_CROSSOVER = true;       // true means every gen
reduce the CrossOver length


// then we can get the current crossove length by this function
int CurtCrossOverLength(int CurtGen){

    double CurtCrossOverLengthRate = IS_REDUCE_CROSSOVER ? ((cGen -
CurtGen)/cGen) * cCrossOverLengthRate : cCrossOverLengthRate;

    return cIndividualLength * CurtCrossOverLengthRate;

}
```

  By doing so, we can controll the length and make it reduces in the future generations

- In the mutaion part:

```
// the ReducePoint type can make the mutaion less in the future
generations
enum MutationType{OnePoint, TwoPoint, ThreePoint, ReducePoint};

// in the mutaion function , we can make the mutaion type changes 1/3
cPopSize and // // make it smaller in the future generations
MutationType CurtMtype = Mtype;

if (CurtMtype == ReducePoint){

    int Threshold = cGen * 0.333;

    switch (CurtGen / Threshold) {
        case 0:{
            CurtMtype = ThreePoint;
            break;
        }
        case 1:{
            CurtMtype = TwoPoint;
            break;
        }
        case 2:{
            CurtMtype = OnePoint;
            break;
        }
        default:
            CurtMtype = OnePoint;
    }
};
```

3. Evaluation the resual

After a series of parameter selection experiments, I finally determined the best parameters for each dataset.

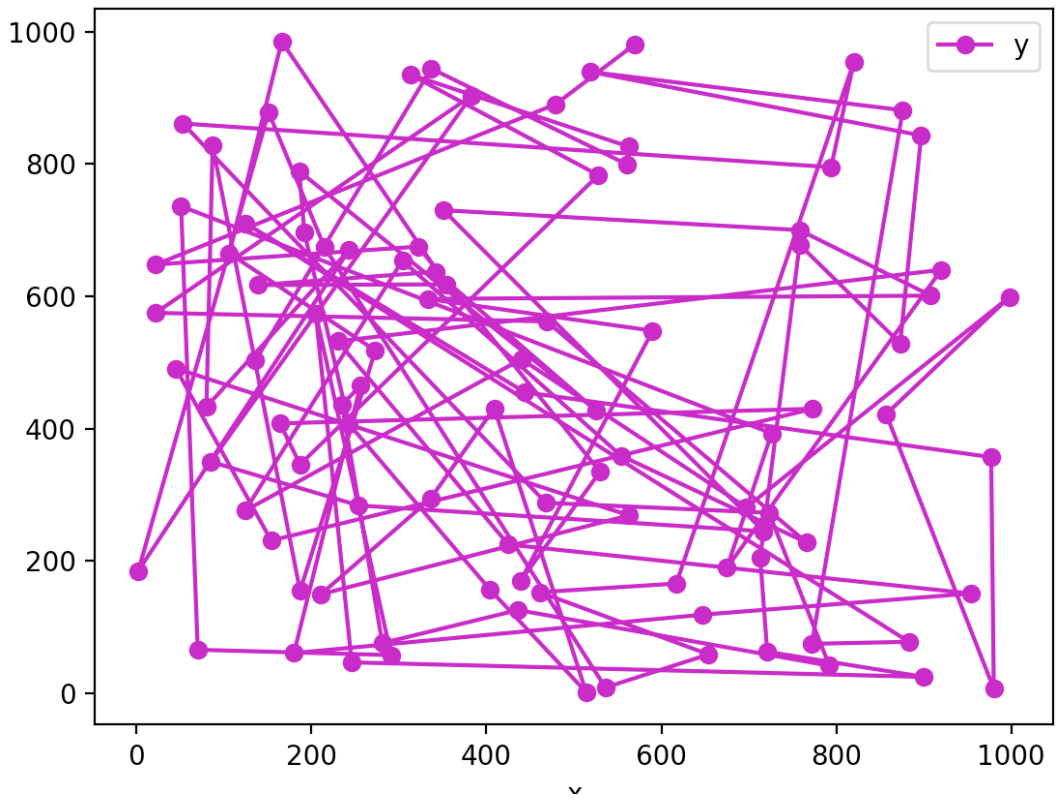
- For dataset tsp100

```
enum MutationType{OnePoint, TwoPoint, ThreePoint, ReducePoint};  //
the type of mutation    // ReducePoint means the mutation reduce after
sevral Gen
enum SelectType{tournament, roulettewheel};          // the type of
Selection Winner
vector<City> AllCity;                                 // store all the
city
int cIndividualLength;                               // length of DNA
which is the number of city
```

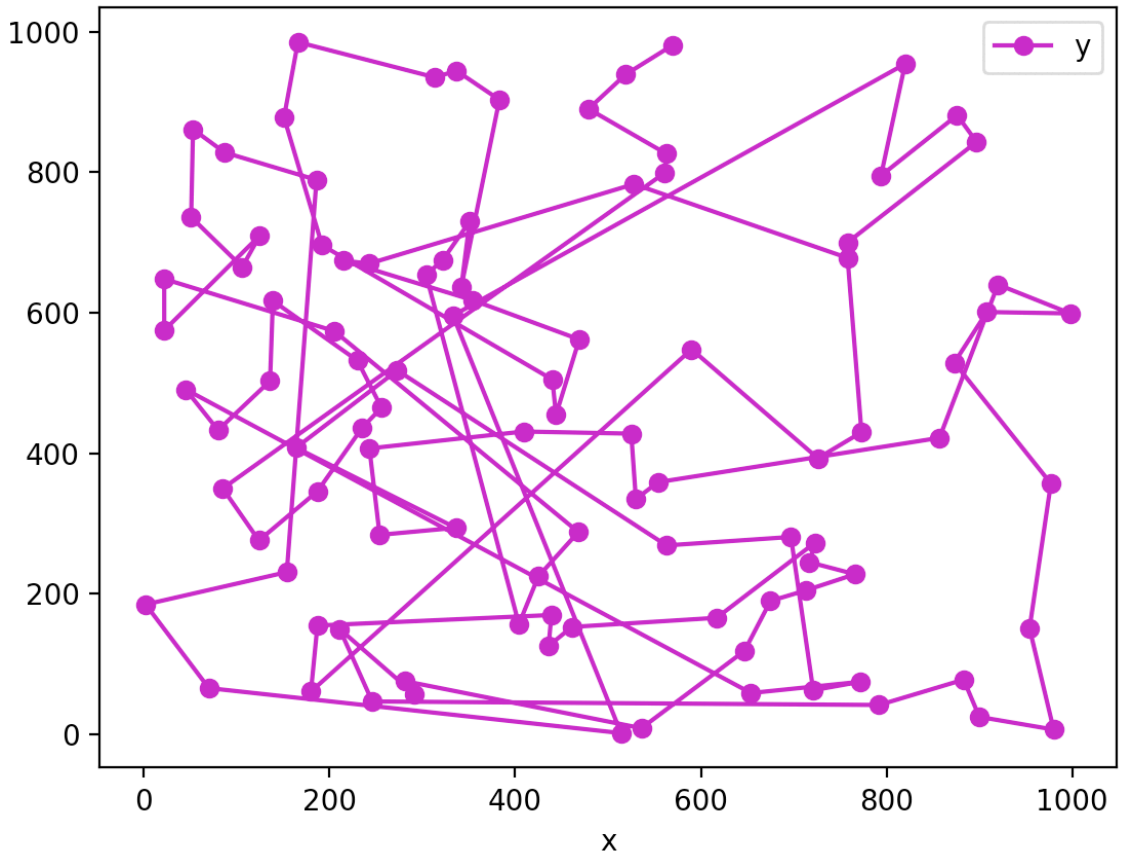
```
const int cPopSize = 150;                                    // must be a even
number cause the cross over
const int cGen = 20000;                                       // how many
round should be taken
const int Seed = 321;                                        // seed for
random number
const int cTournamentSize = 7;                               // the round of
tournament
const double cCrossoverRate = 0.75;
const double cMutationRate = 0.1;
const SelectType SType = tournament;          // define the type of
selection
const MutationType Mtype = ReducePoint;          // define the type of
mutation
const double cCrossOverLengthRate = 0.75;    // the length need to be
cross
map<pair<int, int>, int> type_cost;          // store the mapping
between the different type with the diff cost
map<pair<int, int>, double> look_up;          // cache to store the
comptued distance
const bool IS_WRITE = true;                   // if record the res to a
file
const bool IS_REDUCE_CROSSOVER = true;        // true means every gen
reduce the
```

We can plot the path below:

- after initialize :



- after training :



Obviously the path is clear, Best path in this experiment is:

```
90 86 98 17 57 5 68 0 65 83 7 1 38 92 81 47 34 10 9 50 72 15 3 56 8 73 87
21 78 85 74 23 24 45 33 84 53 31 71 20 49 29 76 40 16 19 48 96 75 97 79 94
41 95 6 25 66 51 2 62 37 80 63 67 32 12 91 44 4 89 11 39 35 43 58 52 82 27
13 69 18 77 61 30 22 60 55 14 42 26 36 93 46 70 54 99 64 88 28 59

with fitness :
49091.5
```

- For dataset tsp200 :

```
enum MutationType{OnePoint, TwoPoint, ThreePoint, ReducePoint};  //
the type of mutation    // ReducePoint means the mutation reduce after
sevral Gen
enum SelectType{tournament, roulettewheel};          // the type of
Selection Winner
vector<City> AllCity;                                 // store all the
city
int cIndividualLength;                               // length of DNA
which is the number of city
```

```
const int cPopSize = 150;                                   // must be a even
number cause the cross over
const int cGen = 20000;                                       // how many
round should be taken
const int Seed = 321;                                       // seed for
random number
const int cTournamentSize = 7;                              // the round of
tournament
const double cCrossoverRate = 0.75;
const double cMutationRate = 0.1;
const SelectType SType = tournament;          // define the type of
selection
const MutationType Mtype = ReducePoint;          // define the type of
mutation
const double cCrossOverLengthRate = 0.75;    // the length need to be
cross
map<pair<int, int>, int> type_cost;          // store the mapping
between the different type with the diff cost
map<pair<int, int>, double> look_up;           // cache to store the
comptued distance
const bool IS_WRITE = false;                     // if record the res to
a file
const bool IS_REDUCE_CROSSOVER = true;         // true means every gen
reduce the
```

we get the best fitness:

- For dataset tsp500 :

```
enum MutationType{OnePoint, TwoPoint, ThreePoint, ReducePoint};  //
the type of mutation    // ReducePoint means the mutation reduce after
sevral Gen
enum SelectType{tournament, roulettewheel};          // the type of
Selection Winner
vector<City> AllCity;                                 // store all the
city
int cIndividualLength;                               // length of DNA
which is the number of city
const int cPopSize = 150;                            // must be a even
number cause the cross over
const int cGen = 2000;                                // how many
round should be taken
const int Seed = 321;                                // seed for
random number
const int cTournamentSize = 7;                       // the round of
tournament
const double cCrossoverRate = 0.75;
const double cMutationRate = 0.1;
const SelectType SType = tournament;        // define the type of
selection
const MutationType Mtype = ReducePoint;        // define the type of
mutation
const double cCrossOverLengthRate = 0.75;   // the length need to be
cross
map<pair<int, int>, int> type_cost;        // store the mapping
between the different type with the diff cost
map<pair<int, int>, double> look_up;         // cache to store the
comptued distance
const bool IS_WRITE = false;                  // if record the res to
a file
const bool IS_REDUCE_CROSSOVER = true;       // true means every gen
reduce the
```

we get the best fitness

```
Gen: 1994 BestFintness: 401359
Gen: 1995 BestFintness: 401359
Gen: 1996 BestFintness: 400532
Gen: 1997 BestFintness: 400532
Gen: 1998 BestFintness: 400532
Gen: 1999 BestFintness: 400532
Program ended with exit code: 0
```

Part 2 :(Deep Learning)

1. Introducation on the MNIST

The MNIST data set is a handwritten data set. each image is gray image by 28 * 28 with a handwritten number(0 - 9)

Composed of four parts, respectively:

- t10k-labels-idx1-ubyte : test image label
- train-images-idx3-ubyte : train image
- train-labels-idx1-ubyte : train image label
- t10k-images-idx3-ubyte : test image

These data sets are not common image formats after decompression, but data sets in binary format.

These data sets consist 60000 training sample and 10000 test sample. then that we can use it we train and test our deep learning model

2. Train MINST with multinomial logistic regression model

2.1 model definition

In the logistic regression by using the tensorflow we need to predefine each node of the model graph. In

- **In the input layer**, Our input data is a gray image with size of 28 * 28, which can be seen as the one vector of which the length 784, we use the placeholder to point out the the node size which will be used to the data flow

```
x = tf.placeholder(tf.float32, [None, 784]) # mnist data image of
shape 28*28=784
```

- **In the output layer**, out classification number will be 10 cause the number in the inamge will only appear in 0 - 9 which means that we have 10 classification

```
y = tf.placeholder(tf.float32, [None, 10]) # 0-9 digits recognition =>
10 classes
```

Then after define the size of input and output layer, we need we **define the weight and bais vector**. Therefore we need the 784 * 10 weight vector 784 indicates the size of the image

```
# Set model weights
W = tf.Variable(tf.zeros([784, 10]))
b = tf.Variable(tf.zeros([10]))
```

Then we construct the model

```
pred = tf.nn.softmax(tf.matmul(x, W) + b) # Softmax
```

The Softmax function is a generalization of logic functions. It can "compress" a vector containing any real number of K dimensions into another K-dimensional real vector such that each element has a range between (0, 1) and the sum of all elements is 1.

Then we can define the cost which we need to reduce and the optimizer we choose

```
# Minimize error using cross entropy
cost = tf.reduce_mean(-tf.reduce_sum(y*tf.log(pred), reduction_indices=1))
# we use Gradient Descent
optimizer = tf.train.GradientDescentOptimizer(learning_rate).minimize(cost)
```

2. 2 Evaluation

After define the 'node' (method) we use in tensorflow graph, we can start train the model by run the tensorflow session

By use the parameters :

```
learning_rate = 0.01
training_epochs = 25
batch_size = 100
```

we can get the reualt of

```
Epoch: 0017 cost= 0.354868955
Epoch: 0018 cost= 0.351459106
Epoch: 0019 cost= 0.348305490
Epoch: 0020 cost= 0.345389097
Epoch: 0021 cost= 0.342744492
Epoch: 0022 cost= 0.340283010
Epoch: 0023 cost= 0.337937056
Epoch: 0024 cost= 0.335705731
Epoch: 0025 cost= 0.333691080
Optimization Finished!
Accuracy: 0.9139
```

By use the parameters :

```
learning_rate = 0.1
training_epochs = 30
batch_size = 100
```

we can get the reualt of

```
Epoch: 0027 cost= 0.264539101
Epoch: 0028 cost= 0.263774711
Epoch: 0029 cost= 0.263274808
Epoch: 0030 cost= 0.262403277
Optimization Finished!
Accuracy: 0.924
```

which is a litter bit improved compared to the preview one

By observing, using a larger learning rate, we can make the cost drop faster, but the higher the learning rate, the better, because the larger learning rate makes it difficult for our model to reach the global minimum.

3. Train MINST with Convolutional neural network model

3.1 model definition

In this structure, we define a four-layered sentence neural network.

In the beginning, we define the x and y by the placeholder

xs indicates the size of the input image which is the 784 vector and the number is defined as none to load a uncertain number of images

ys indicate the label of the numebr classifications which should be 10

```
xs = tf.placeholder(tf.float32, [None, 784])
ys = tf.placeholder(tf.float32, [None, 10])
```

- Layer one :

  we apply 32 5x5 filters (extract 5x5 pixel sub-areas) and apply the ReLU activation function and use the max_pooing method as our pooling method

  ```
  W_conv1 = weigth_variable([5, 5, 1, 32])
  b_conv1 = weigth_variable([32])
  # cnv
  h_conv1 = tf.nn.relu(conv2d(x_image, W_conv1) + b_conv1)  # 28*28*32
  # pooing
  h_pool1 = max_pool_2x2(h_conv1)  # 14*14*32
  ```

- layer two :

  we apply 64 5x5 filters and apply the ReLU activation function, and also use the max_pooing method as our pooling method

```
W_conv2 = weigth_variable((5, 5, 32, 64))
b_conv2 = weigth_variable([64])
h_conv2 = tf.nn.relu(conv2d(h_pool1, W_conv2) + b_conv2)  # 14*14*64
h_pool2 = max_pool_2x2(h_conv2)  # 7*7*64
```

- layer three:

  Define the third layer of fully connected layers, which contains 1024 neurons

  and we can get the size of weigth by computing the pooling which in layer two

  ```
  W_fc1 = weigth_variable([7 * 7 * 64, 1024])
  b_fc1 = bias_varibale([1024])
  h_pool2_flat = tf.reshape(h_pool2, [-1, 7 * 7 * 64])
  h_fc1 = tf.nn.relu(tf.matmul(h_pool2_flat, W_fc1) + b_fc1)
  # aviod the overfit
  h_fc1_drop = tf.nn.dropout(h_fc1, keep_prob)
  ```

- Layer four:

  which is the last layer and it contain 10 neural which corresponding to the 10 classification if the handwritten numebr type

  ```
  W_fc2 = weigth_variable([1024, 10])
  b_fc2 = bias_varibale([10])
  ```

Then we use the Logistic regression which we use in the task one, to computer each number's probability

```
prediction = tf.nn.softmax(tf.matmul(h_fc1_drop, W_fc2) + b_fc2)
```

Then we define the loss function which we use the **cross entropy**, and define the **Optimizer as the Adam**,

```
cross_entropy = tf.reduce_mean(-tf.reduce_sum(ys * tf.log(prediction),
reduction_indices=[1]))
train_step = tf.train.AdamOptimizer(1e-4).minimize(cross_entropy)
```

Till now, we have build our model successfully.

3.2 Evalution

After we define the model, then we can use the tensorflow seesion to load the data, we use the train enpoch as 1000 times, and the batch as 100:

```
ENPOCE = 1000
batch = 100
```

```
for epoce in range(ENPOCE):
    batch_xs, batch_ys = mnist.train.next_batch(batch)
    sess.run(train_step, feed_dict={xs: batch_xs, ys: batch_ys, keep_prob:
0.5})
    if epoce % 50 == 0:
        accuracy = compute_accuracy(
            mnist.test.images[:1000], mnist.test.labels[:1000])
    print("epoch: %d  acc: %f" % (epoce + 1, accuracy))
```

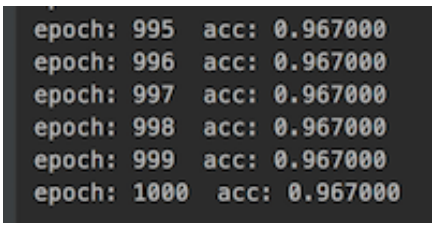
and get the result :

```
epoch: 993  acc: 0.971000
epoch: 994  acc: 0.971000
epoch: 995  acc: 0.971000
epoch: 996  acc: 0.971000
epoch: 997  acc: 0.971000
epoch: 998  acc: 0.971000
epoch: 999  acc: 0.971000
epoch: 1000  acc: 0.971000
```

- Then i set the 3 * 3 kernel size which is also a norm when using the convolutional network in layer one :

```
W_conv1 = weigth_variable([7, 7, 1, 32])
```

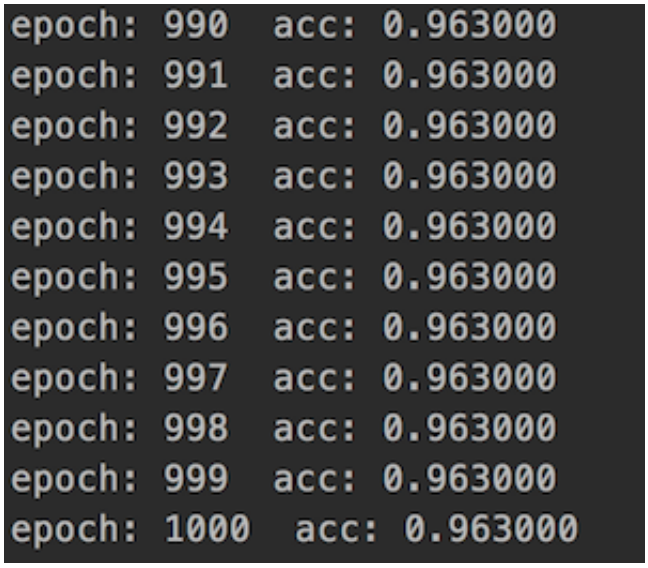
However the result seems not improved



- Then i use the 3 * 3 kernel size to generate the convolutional network in layer one :

```
W_conv1 = weigth_variable([7, 7, 1, 32])
```

And get the resualt below:

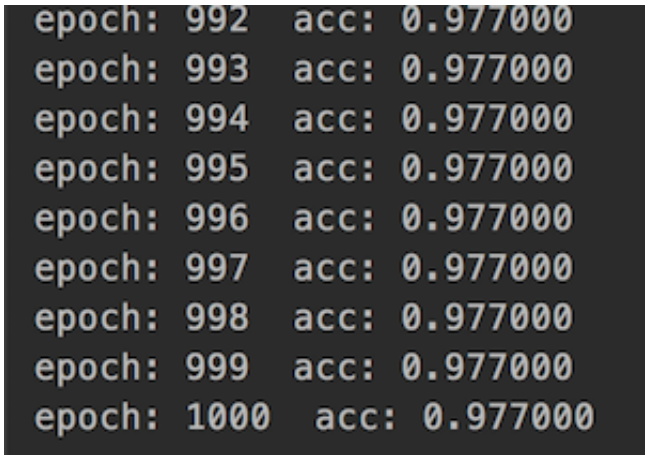


After observation, choosing a 5*5 kernel seems to be a better choice, it has higher accuracy.

- Then i change the batch size to 200 which can make the speed become much more slower

```
ENPOCE = 1000
batch = 300
```

  then we can get the result below which seems imporved a litter bit:



Batch only defines the size of the training data in a batch, and there is no direct correlation with the accuracy of the model, but the larger the batch, the larger the time will be.

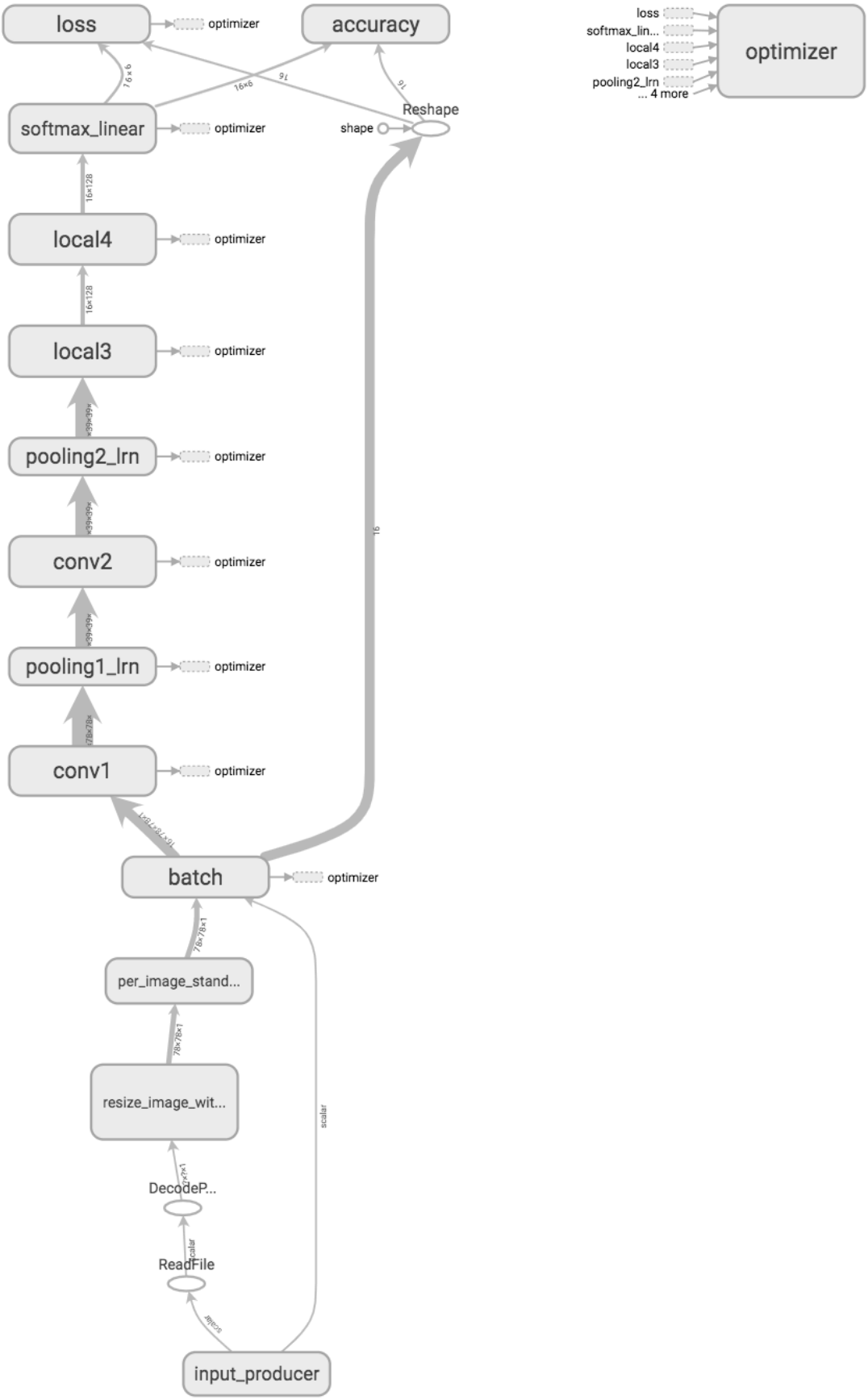
4. Train Cell-Image with Convolutional neural network

4.1 basic technique i use in this task

Due to the complexity of cell image processing, I chose deep learning in the training model, and the excellent performance on CNN processing images in task2. I decide that **using CNN-based deep neural network** to complete task3, In the choose of toolbox I use the **Tensorflow** as a depp learning library and us the **python** as the language which has much support in tensorflow compared to the C++ etc. Also i use the **pandas, numpy** to do some martrix compute and **matplotlib** to visualize the data

4.2 Structure of the classification system

I use the **tensorborad** to plot the structure i use in this CNN system , Similar to task2, our CNN structure has a 4-layer network, in which the first two layers are convolutional layers, doing cvn and pooling work, and the latter two layers are full-link layers.In the data input part, we use the queue to read the data. When the amount of data reaches a certain level, it can speed up the reading and writing of data, greatly increasing the efficiency. In terms of prediction, I used logistic regression to process the results, and I could never settle the specific probability of the image for a certain classification.



4.3 prepare the image data

- Standardize each image

```
image = per_image_standardization(image)
```

Linearly scales `image` to have zero mean and unit variance.

This op computes `(x - mean) / adjusted_stddev`, where `mean` is the average of all values in image, and `adjusted_stddev = max(stddev, 1.0/sqrt(image.NumElements()))`.

`stddev` is the standard deviation of all values in `image`. It is capped away from zero to protect against division by 0 when handling uniform images.

- resize the image

  Since the size of the input image may be inconsistent. In order to ensure the correctness of the matrix operation, we resize the image. This process can make the size of each image read consistent. If it is too large, it will be cropped. If it is too small, it will be filling

```
# image_widtd = 78
# image_height = 78
image = tf.image.resize_image_with_crop_or_pad(image, image_width,
image_height)
```

- Relabel the image

  Since the labels of all images are in the form of characters, this form will make the data unsuitable, so we changed the label of each image from a string to an int, and one-to-one correspondence.

```
Image_class = {
    'Speckled': 0,
    'Centromere': 1,
    'Nucleolar': 2,
    'Homogeneous': 3,
    'NuMem': 4,
    'Golgi': 5
}
 data_label['Image class'] = data_label['Image class'].apply(lambda x:
Image_class[x])
```

4.3 Evaluation

4.3.1 parameter choose

I chose the following two sets of parameters for experimentation. The main difference is the learning rate and the batch size

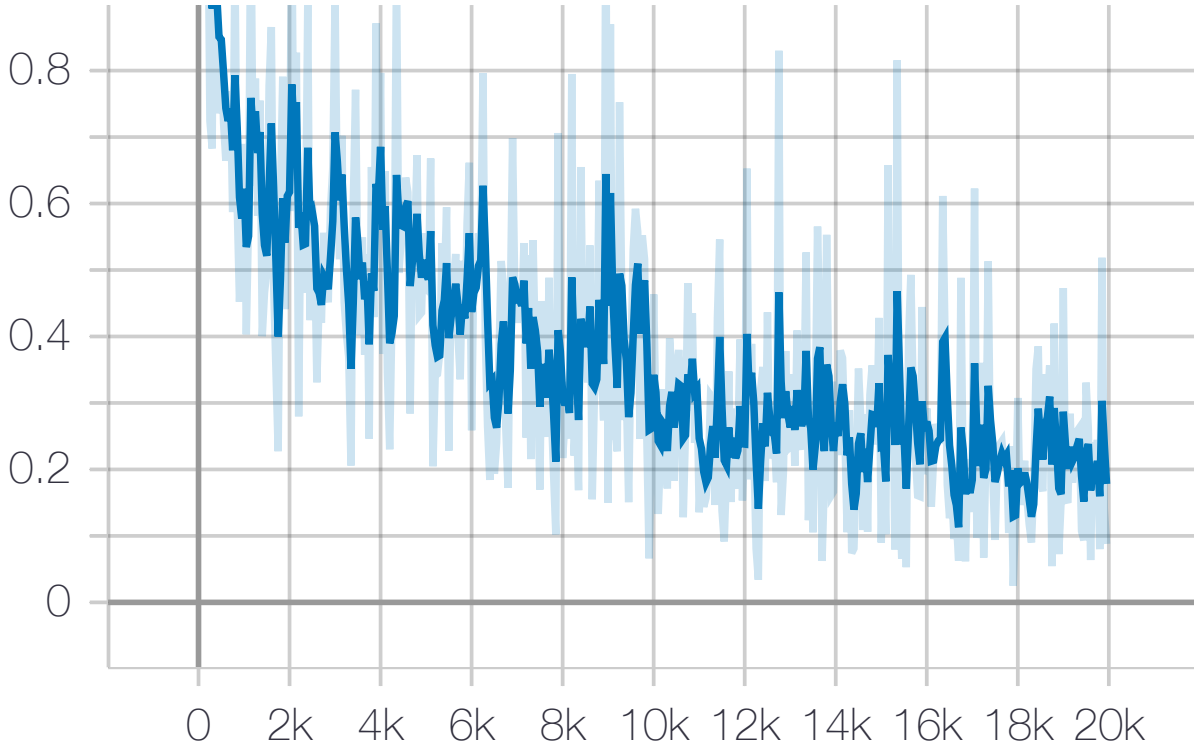
```
N_CLASSES = 6
IMG_W = 78              # resize the image, if the input image is too large,
training will be very slow.
IMG_H = 78
BATCH_SIZE = 16
CAPACITY = 2000
MAX_STEP = 20000         # it is suggested to use MAX_STEP>10k
learning_rate = 0.0001  # it is suggested to use learning rate<0.0001
```

```
N_CLASSES = 6
IMG_W = 78              # resize the image, if the input image is too large,
training will be very slow.
IMG_H = 78
BATCH_SIZE = 30
CAPACITY = 2000
MAX_STEP = 20000         # it is suggested to use MAX_STEP>10k
learning_rate = 0.00005  # it is suggested to use learning rate<0.0001
```

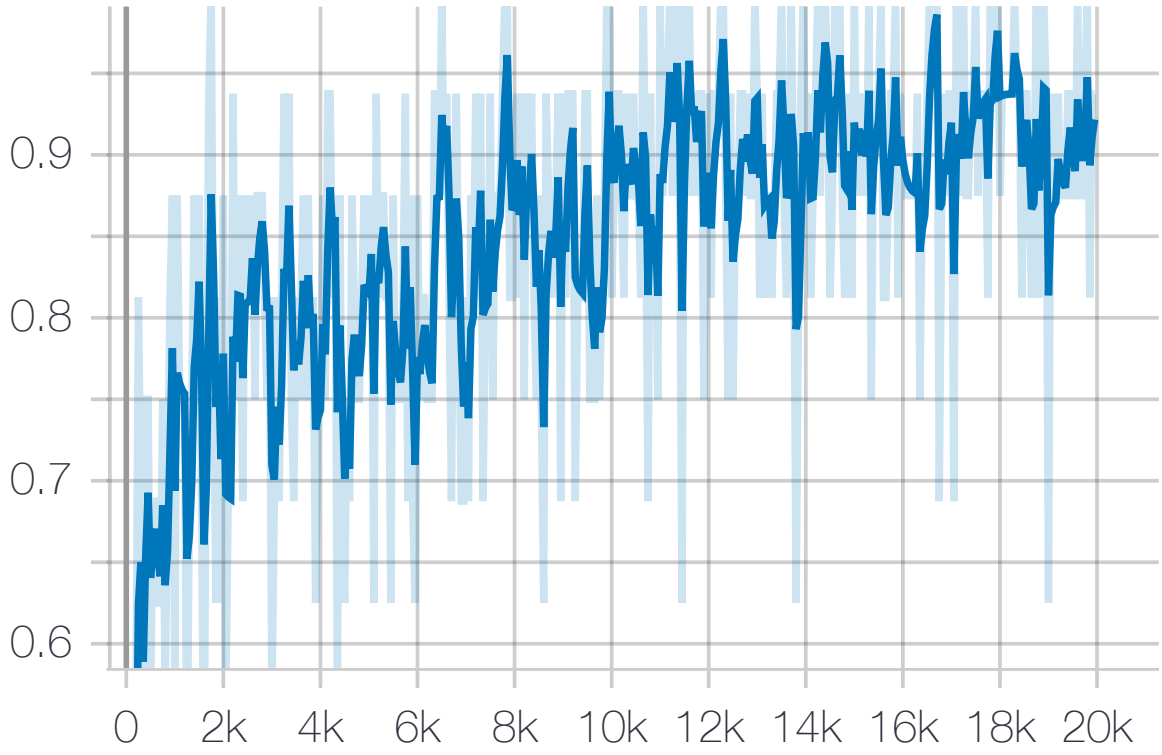
4.3.2 loss and the accuracy plot

- Fot the first set of the parameter:

  Loss

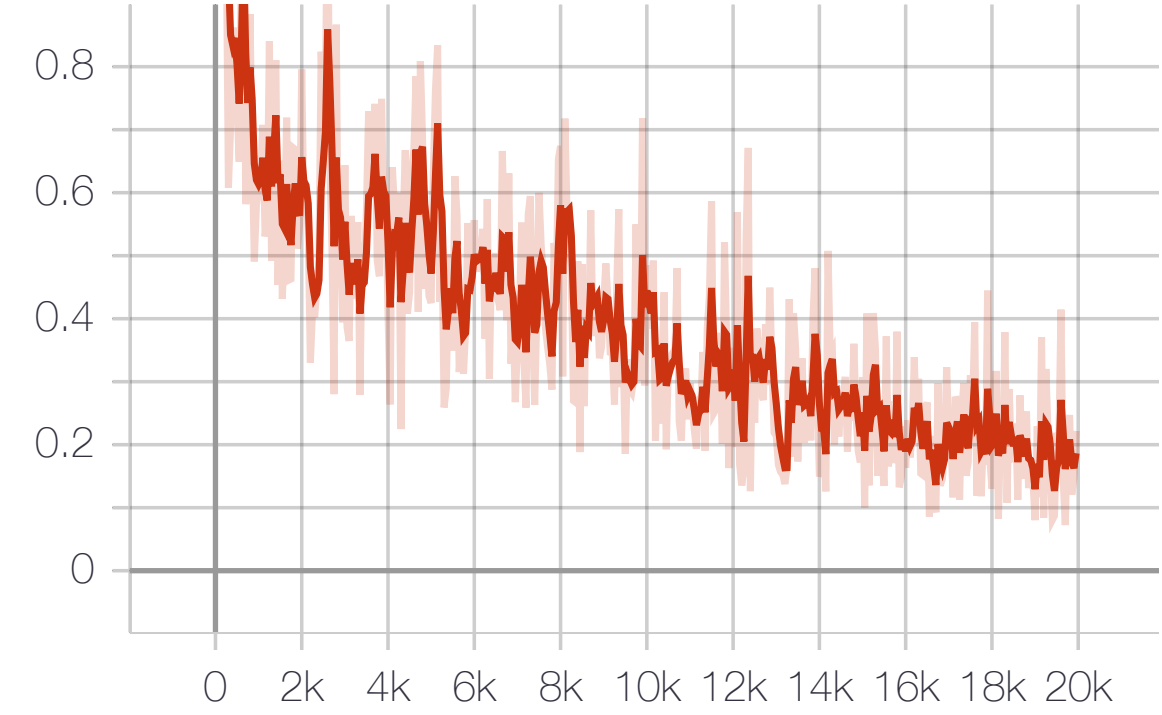


accuracy

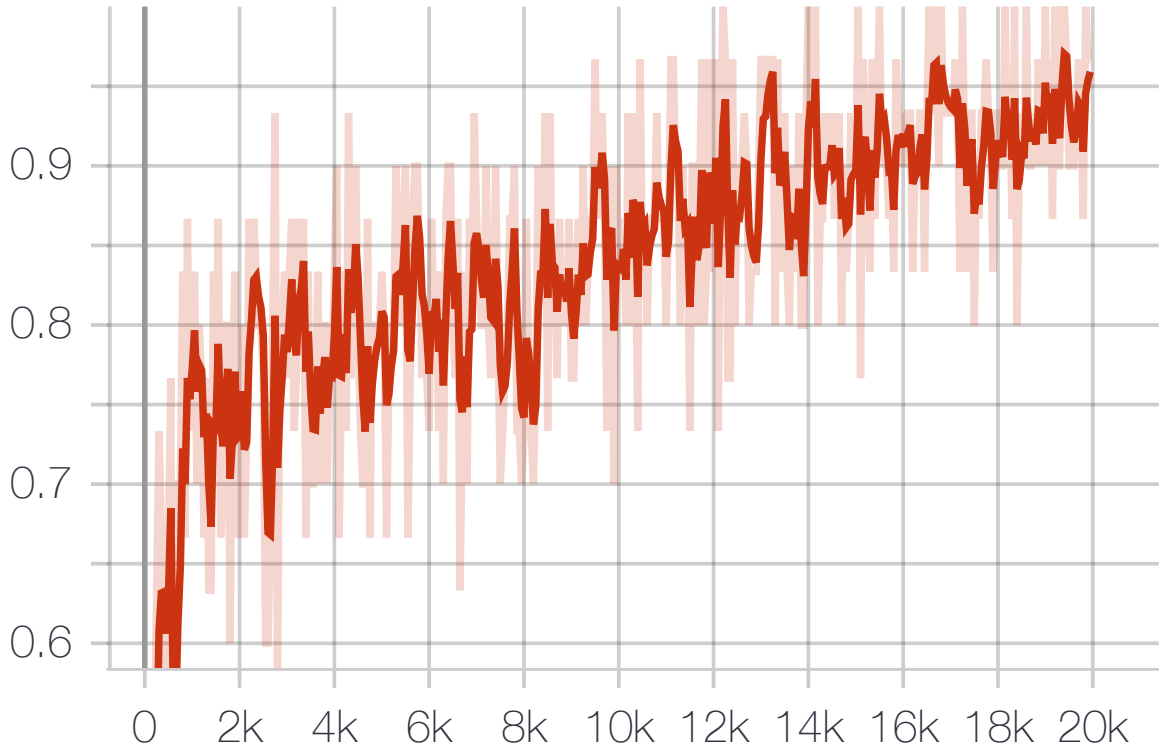


- Fot the second set of the parameter:

  Loss



  accuracy



4.3.3 Final accuracy on the test set

- Fot the first set of the parameter:

```
Loading success, global_step is 19999
------------------------
The model's loss is 0.47
Correct : 2324
Wrong : 396
The accuracy in test images are 85.45%
```

- Fot the second set of the parameter:

```
The model's loss is 0.48
Correct : 2280
Wrong : 440
The accuracy in test images are 83.84%
```

4.3.4 Improved the accuracy

Through analysis, the most accurate situation is that the accuracy is not enough which may be resualt by the amount of data is not enough. In this example, there are less than 10,000 training data. This is obviously not enough for training a high-performance convolutional neural network. Therefore, we can Try to use **data argumentation** to increase the accuracy for this problem

Tenserflow has offer the library to do the data argumentation, such as **flip, crop, rotate** a image which can increase the amount of image without change the label.

```
# crop
crop_img = tf.random_crop(img,[width, heigth, chanel])
# flip
h_flip_img = tf.image.random_flip_left_right(img)
v_flip_img = tf.image.random_flip_up_down(img)
```

In this way, we can get more data to train a more accurate model.

4.4 Summary

In terms of training speed: CNN training speed is significantly faster than some fully-linked neural networks such as ANN, which is due to the fact that the first two layers of CNN are not fully linked neural networks, and have entered the CNV and pooling layers. The dimension of the input data is greatly reduced, thereby increasing the training speed. Compared with the model of non-neural network architecture, CNN has the key advantage of neural network, that is, no manual manual selection feature is needed. However CNN needs to adjust the parameters, and at the same time, it needs a large sample size to achieve the good effect.Therefore, in the face of insufficient data, you can use data argumentation to greatly increase the amount of data, so as to achieve a good training effect.