

My First Insight into Machine Learning

Keith Null
May 6, 2018

Contents

1	Problem Description	2
1.1	Exercise 1	2
1.2	Exercise 2	2
1.3	Exercise 3	2
1.4	Exercise 4 (Optional)	2
2	Problem Analysis	3
2.1	Skills Involved	3
2.2	Solution Design	3
3	Common Part: Extracting Features	4
3.1	Getting Data From Database	4
3.2	Processing Data With Python	4
4	Solution One: Using Scikit-learn	6
4.1	Loading Data	6
4.2	Logistic Regresssion	7
4.3	Naive Bayes	7
4.4	SVM	8
4.4.1	Different Kernels	8
4.4.2	Optimizing More Parameters	9
4.5	Comparasion	9
5	Solution Two: Using Tensorflow	10
5.1	DNN Classifier	10
5.1.1	Loading Data	10
5.1.2	Training and Testing	10
5.1.3	Result	11
5.2	CNN Classifier	11
5.2.1	Architecture of my CNN	11
5.2.2	Loading Data	13
5.2.3	Training and Testing	13
5.2.4	Some Hyper-parameters	14
5.2.5	Result	14
6	Conclusion And Prospect	15

1 Problem Description

1.1 Exercise 1

Implement a function, which takes IDs of two scholars that once worked together as parameters and returns features about the cooperation relationship between them.

Nine possible features are given for reference.

1.2 Exercise 2

Download training and testing files. Then extract features of two scholars in the same line in the training file to construct X matrix and use given labels to construct y vector. With X and y, train a `LogisticRegression()` model in `sklearn` and evaluate it with the data in the testing file.

1.3 Exercise 3

Similar to Exercise 2, try to use SVM and Naive Bayes models. Also, try SVM models with different kernels.

Make a comparison between them concerning their training time and prediction accuracy.

1.4 Exercise 4 (Optional)

Implement a simple neural network classifier with Tensorflow and compare it with the previous traditional classifiers.

2 Problem Analysis

2.1 Skills Involved

Firstly, some basic knowledge of SQL is needed to get data from the database. Then, I need to use Python to handle these data. As for the part of machine learning, the usage of numpy, sklearn and tensorflow is required.

Beyond these, an insight into how SVM and CNN work would also help.

2.2 Solution Design

Just like the common procedure of machine learning, extracting features, loading data, training a model and evaluating the model are steps of what I'm going to do.

Whatever machine learning model I choose, extracting features is the common part before training and testing.

Then, for traditional machine learning, I can use Scikit-learn, a useful machine learning package of Python; for deep learning, Tensorflow provides a relatively simple way for me to do the job.

3 Common Part: Extracting Features

To better organize codes, a class named `FeatureExtractor()` is defined in `feature.py`, which contains methods like connecting to the database and extracting features. The main parts of it are illustrated as follows:

3.1 Getting Data From Database

First, when an instance of `FeatureExtractor()` is initialized, the connection to the database is not established.

```
1 def __init__(self):
2     self.connected = False
```

Then the `connect()` method is to be called to establish a connection.

```
1 def connect(self, user, password, db,
2             host="localhost", port=3306, charset="utf8"):
3     try:
4         self.connection = pymysql.connect(host=host, user=user,
5                                           password=password, db=db,
6                                           port=port, charset=charset)
7         self.cursor = self.connection.cursor()
8     except:
9         print("Failed to connect to the database!")
10    else:
11        self.connected = True
```

For convenience, I just get all the papers of the two scholars in the order of publishing year from the database. The SQL query is like this:

```
1 SELECT paper_author_affiliation.*,papers.paperpublishyear
2 FROM paper_author_affiliation,papers
3 WHERE paper_author_affiliation.paperid=papers.paperid
4     AND ( paper_author_affiliation.authorid="00000000"
5           OR paper_author_affiliation.authorid="11111111")
6 ORDER BY papers.paperpublishyear ASC,
7         paper_author_affiliation.paperid;
```

3.2 Processing Data With Python

In Python, this query's result is a two dimensional list, of which each element is a list containing PaperID, AuthorID, AffiliationID, AuthorSequence and PaperPublishYear. The problem is that the same paper may occur twice in the result if it's done by the two scholars together. So, I need to process the data by myself.

First, the query's result needs to be re-organized. My solution is as follows:

```
1 def get_all_papers(self, author1, author2):
2     if not self.connected:
3         raise RuntimeError("Haven't connected to the database yet!")
4     query = "..." #omitted here
5     try:
6         self.cursor.execute(query)
7     except:
8         print("Something unexpected happened when handling the SQL query:")
9         print(query)
10        self.connection.rollback()
11        return ()
12    else:
13        raw_result = self.cursor.fetchall()
```

```

14     result = list()
15     for row in raw_result:
16         if len(result) == 0 or row[0] != result[-1][0]:
17             paper_info = [row[0], row[3] if row[1] == author1 else 0,
18                           row[3] if row[1] == author2 else 0, row[4]]
19             result.append(paper_info)
20         else:
21             if row[1] == author1:
22                 result[-1][1] = row[3]
23             else:
24                 result[-1][2] = row[3]
25     return result

```

After that, each element of the result is a list of PaperID, IsAuthor1, IsAuthor2, PaperPublishYear.

Then, extract features from the result. Though the given nine features might not be the best, but as far as I'm concerned, they are reasonable enough. Therefore, I just extract these nine features.

When doing this, I make use of some Python features like list comprehension and set to simplify the code:

```

1  def extract_feature(self, author1, author2):
2      all_papers = self.get_all_papers(author1, author2)
3      assert all_papers, "Error! No paper."
4      feature = list()
5      cooperations = [i for i in range(
6          len(all_papers)) if all_papers[i][1] and all_papers[i][2]]
7      papers_of_author1 = [i for i in range(
8          len(all_papers)) if all_papers[i][1]]
9      papers_of_author2 = [i for i in range(
10         len(all_papers)) if all_papers[i][2]]
11     feature.append(
12         len([i for i in range(cooperations[0]) if all_papers[i][1]]))
13     feature.append(
14         len([i for i in range(cooperations[0]) if all_papers[i][2]]))
15     feature.append((feature[0]-feature[1])/len(cooperations))
16     feature.append(all_papers[cooperations[0]]
17                   [3]-all_papers[papers_of_author1[0]][3])
18     feature.append(all_papers[cooperations[0]]
19                   [3]-all_papers[papers_of_author2[0]][3])
20     feature.append(
21         ( feature[3]-feature[4] ) / ( all_papers[cooperations[-1]][3] -
22         ↪ all_papers[cooperations[0]][3] + 1))
23     feature.append( len ( [i for i in
24         ↪ set(papers_of_author1).difference(set(papers_of_author2)
25         ↪ if all_papers[cooperations[0]][3] <= all_papers[i][3] <=
26         ↪ all_papers[cooperations[-1]][3]])
27
28     feature.append( len ( [i for i in
29         ↪ set(papers_of_author2).difference(set(papers_of_author1)
30         ↪ if all_papers[cooperations[0]][3] <= all_papers[i][3] <=
31         ↪ all_papers[cooperations[-1]][3]])
32
33     feature.append((feature[6]-feature[7])/len(cooperations))
34     return feature

```

Nearly finished but one thing to note is that the database connection must be closed properly after extracting features. So for this class, a destructor is needed.

```

1  def __del__(self):
2      if self.connected:
3          self.connection.close()

```

4 Solution One: Using Scikit-learn

4.1 Loading Data

For different models in Sklearn, their usage is quite similar. And the part of loading data is the same for following models. As the loading of training data and testing data has no difference, a function `load_data()` is implemented.

When `load_data()` is called, it first checks whether the features have been extracted and saved before. If so, it just loads features from features files by calling `load_from_feature_file()`. If not, it calls `load_from_raw_file()`, which extracts features and saves them into certain files for future use. So the code is as follows:

```
1 @timer #attention here
2 def load_from_raw_file(filepath, feature_extractor, save_into_file=None):
3     with open(filepath, mode="r", encoding="utf8") as file:
4         X, y = list(), list()
5         i = 1
6         for line in file:
7             print(i, end="\r")
8             i += 1
9             author1, author2, relation = line.strip().split()
10            features = feature_extractor.extract_feature(
11                author1, author2)
12            X.append(features)
13            y.append(int(relation))
14        if(save_into_file):
15            with open(save_into_file, mode="w", encoding="utf8") as saving_file:
16                for feature in zip(y, X):
17                    saving_file.write(
18                        str(feature[0])+" "+" ".join(map(str, feature[1]))+"\n")
19        return X, y

20 @timer #attention here
21 def load_from_feature_file(filepath):
22     with open(filepath, mode="r", encoding="utf8") as file:
23         X, y = list(), list()
24         i = 1
25         for line in file:
26             print(i, end="\r")
27             i += 1
28             temp_list = line.strip().split()
29             y.append(int(temp_list[0]))
30             X.append(list(map(float, temp_list[1:])))
31        return X, y

32 def load_data():
33     try:
34         extractor = feature.FeatureExtractor()
35         extractor.connect("root", "", "academicdb")
36     except:
37         connected = False
38     else:
39         connected = True
40     if os.path.exists("../Data/train_feature.txt"):
41         print("Start to load training data from feature file.")
42         X_train, y_train = load_from_feature_file("../Data/train_feature.txt")
43     else:
44         assert connected
45         print("Start to load training data from raw file.")
46         X_train, y_train = load_from_raw_file(
```

```

47         "../Data/train.txt", extractor,
48         ↪ save_into_file="../Data/train_feature.txt")
49 if os.path.exists("../Data/test_feature.txt"):
50     print("Start to load testing data from feature file.")
51     X_test, y_test = load_from_feature_file("../Data/test_feature.txt")
52 else:
53     assert connected
54     print("Start to load testing data from raw file.")
55     X_test, y_test = load_from_raw_file(
56         "../Data/test.txt", extractor, save_into_file="../Data/test_feature.txt")
57 return X_train, y_train, X_test, y_test

```

By the way, as you might have noticed, I also implement a decorator named timer to measure how fast or how slowly a function runs. This decorator can be reused when training models.

```

1 import time
2 import functools
3 def timer(func):
4     @functools.wraps(func)
5     def wrapper(*args, **kw):
6         tic = time.time()
7         result = func(*args, **kw)
8         tok = time.time()
9         print("Runtime:{:.3f}s".format(tok-tic))
10        return result
11    return wrapper

```

For example, when loading training data, `load_from_feature_file()` takes less than 1 second while `load_from_row_file()` consumes nearly 1 minute. So, it's quite necessary to save features into files.

4.2 Logistic Regression

As in the sklearn module premade Logistic Regression Model is provided, things are relatively easy. Just need to import it and use it like these:

```

1 from sklearn.linear_model import LogisticRegression
2 @timer
3 def use_logistic_regression(X_train, y_train, X_test, y_test):
4     model = LogisticRegression()
5     print("Start to train a logistic regression model.")
6     model.fit(X_train, y_train)
7     score = model.score(X_test, y_test)
8     print("Score of logistic regression:", score)

```

The output information is :

```

1 Start to train a logistic regression model.
2 Score of logistic regression: 0.75
3 Runtime:0.026s

```

4.3 Naive Bayes

Similar to the use of Logistic Regression Model, just import and use Gaussian Naive Bayes Model (or something else like Bernoulli Naive Bayes).


```

1 from sklearn.naive_bayes import GaussianNB
2 @timer
3 def use_naive_bayes(X_train, y_train, X_test, y_test):
4     model = GaussianNB()
5     print("Start to train a naive bayes model.")
6     model.fit(X_train, y_train)
7     score = model.score(X_test, y_test)
8     print("Score of naive bayes:", score)

```

The output information is :

```

1 Start to train a naive bayes model.
2 Score of naive bayes: 0.720543806647
3 Runtime:0.016s

```

4.4 SVM

The general usage of SVM in sklearn is also roughly the same as the previous models:

```

1 @timer
2 def use_SVM(X_train, y_train, X_test, y_test, kernel="linear"):
3     try:
4         model = SVC(kernel=kernel)
5         print("Start to train a SVM model(kernel: {0}).".format(kernel))
6         model.fit(X_train, y_train)
7         score = model.score(X_test, y_test)
8         print("Score of SVM(kernel: {0}):".format(kernel), score)
9     except:
10        print("Error!")

```

4.4.1 Different Kernels

As we know, SVM has different kernels such as linear, rbf and sigmoid. Using different kernel has an impact on the performance of SVM. So I have tried several kernels in this way:

```

1 SVM_kernels = ["linear", "rbf", "sigmoid"]
2 for kernel in SVM_kernels:
3     use_SVM(X_train, y_train, X_test, y_test, kernel)

```

The result is as follows:

```

1 Start to train a SVM model(kernel: linear).
2 Score of SVM(kernel: linear): 0.730362537764
3 Runtime:6.807s
4
5 Start to train a SVM model(kernel: rbf).
6 Score of SVM(kernel: rbf): 0.690332326284
7 Runtime:2.324s
8
9 Start to train a SVM model(kernel: sigmoid).
10 Score of SVM(kernel: sigmoid): 0.615558912387
11 Runtime:1.207s

```

4.4.2 Optimizing More Parameters

For SVM, there're quite a few parameters to adjust more than just kernel. Among them, c and γ are two critical parameters. But how to set them to a reasonable value? By cross validation. In sklearn, GridSearchCV() class can help me do th job as it searches for the best given values of parameters. The code is like these:

```
1 def optimize_SVM(X_train, y_train, X_test, y_test):
2     C_range = np.logspace(-4, 3, 8)
3     gamma_range = np.logspace(-7, 0, 8)
4     kernel_range = ["linear", "rbf"]
5     param_grid = dict(gamma=gamma_range, C=C_range, kernel=kernel_range)
6     grid = GridSearchCV(SVC(),
7                         param_grid=param_grid, n_jobs=-1,)
8     grid.fit(X_train[:100], y_train[:100])
9     print("The best parameters are {0} with a score of {1}".format(grid.best_params_,
    ↪ grid.best_score_))
```

Note that I just take the first 100 groups of features in training data, as too much data would consume an unaffordable amount of time. After about 10 minutes, the result is as follows:

```
1 The best parameters are {'C': 10.0, 'gamma': 0.0001, 'kernel': 'rbf'} with a score of
    ↪ 0.73
```

Then I use the "best" parameters to train a SVM again, the result is like these, slightly improved.

```
1 Start to train a SVM model(kernel: rbf).
2 Score of SVM(kernel: rbf): 0.748489425982
3 Runtime:0.799s
```

Adjusting the range of parameters and re-optimizing might continue to improve the result, but as it would take too much time, I didn't do that.

4.5 Comparasion

As the result shows, different models, even different parameters make a difference on the performance of training and evaluating. It might be a little arbitray for me to make judgements about them, but some phenomena are evident:

First, Logistic Regression and Naive Bayes are both fast enough when training, while SVMs, especially linear kerneled one, are relatively slow.

Second, as for the prediction accuracy with the default parameters in sklearn, Logistic Regression, Naive Bayes and linear kerneled SVM are at the same level, while other SVMs performs a little disappointingly.

However, with relatively proper parameters, SVM with the rbf kernel wins over SVMs with other kernels, but still at the same level as Logistic Regression and Naive Bayes.

5 Solution Two: Using Tensorflow

5.1 DNN Classifier

When starting to use Tensorflow, I first decide to use a Deep Neural Network Classifier. Considering the size of training data and complexity of problem, two or three layers are enough. If there're too much layers, the performance would not improve but deteriorate.

Before starting to explain my solution, I must make it clear that some of my code consults the official tutorial of Tensorflow to avoid misunderstandings.

5.1.1 Loading Data

As Tensorflow has a premade DNN classifier, `tf.estimator.DNNClassifier()`, I just make use of it. Then the loading of data also needs to be adjusted as the classifier takes data in dict.

```
1 def convert_into_dict(X_raw):
2     X = dict()
3     for key in range(len(X_raw[0])):
4         X[str(key)] = np.array([row[key] for row in X_raw])
5     return X
6
7 X_train, y_train, X_test, y_test = load_data.load_data()
8 X_train = convert_into_dict(X_train)
9 X_test = convert_into_dict(X_test)
```

5.1.2 Training and Testing

This part just follows the usual pattern of Tensorflow and the instructions of Tensorflow's tutorial.

```
1 my_feature_columns = []
2 for key in X_train.keys():
3     my_feature_columns.append(tf.feature_column.numeric_column(key=key))
4 # Build 2 hidden layer DNN with 10, 10 units respectively.
5 classifier = tf.estimator.DNNClassifier(
6     feature_columns=my_feature_columns,
7     # Two hidden layers of 10 nodes each.
8     hidden_units=[10, 10],
9     # The model must choose between 2 classes.
10    n_classes=2)
11 # Train the Model.
12 classifier.train(
13     input_fn=lambda: load_data.train_input_fn(X_train, y_train, 200),
14     steps=10000)
15 # Evaluate the model.
16 eval_result = classifier.evaluate(
17     input_fn=lambda: load_data.eval_input_fn(X_test, y_test, 200))
18 print('\nTest set accuracy: {accuracy:0.3f}\n'.format(**eval_result))
```

As for `train_input_fn()` and `eval_input_fn()` in `load_data.py`, I also refer to the tutorial:

```
1 def train_input_fn(features, labels, batch_size):
2     """An input function for training"""
3     # Convert the inputs to a Dataset.
4     dataset = tf.data.Dataset.from_tensor_slices((dict(features), labels))
5     # Shuffle, repeat, and batch the examples.
6     dataset = dataset.shuffle(1000).repeat().batch(batch_size)
7     # Return the dataset.
8     return dataset
```

```

9 def eval_input_fn(features, labels, batch_size):
10     """An input function for evaluation or prediction"""
11     features = dict(features)
12     if labels is None:
13         # No labels, use only features.
14         inputs = features
15     else:
16         inputs = (features, labels)
17     # Convert the inputs to a Dataset.
18     dataset = tf.data.Dataset.from_tensor_slices(inputs)
19     # Batch the examples
20     assert batch_size is not None, "batch_size must not be None"
21     dataset = dataset.batch(batch_size)
22     # Return the dataset.
23     return dataset

```

5.1.3 Result

After 10000 steps of training, which takes roughly 30 seconds, the result is as follows:

```

1 INFO:tensorflow:Saving dict for global step 10000: accuracy = 0.761329,
  ↳ accuracy_baseline = 0.548338, auc = 0.855578, auc_precision_recall = 0.872124,
  ↳ average_loss = 0.465701, global_step = 10000, label/mean = 0.548338, loss =
  ↳ 88.0841, precision = 0.809668, prediction/mean = 0.53995, recall = 0.738292
2 Test set accuracy: 0.761

```

Compared with traditional machine learning methods, DNN takes much more time to train (though in this case it's not very evident due to the small size of data), but has a higher accuracy. (also not very evident for the same reason)

5.2 CNN Classifier

In most cases, Convolutional Neural Network Classifier works better than Deep Neural Network Classifier. Thus I decide to try CNN Classifier for this problem.

5.2.1 Architecture of my CNN

With Tensorflow, a customized CNN Classifier can be constructed for this problem. Also, considering the size of training data and the complexity of the problem, too much layers are useless. So the architecture of my CNN is like these:

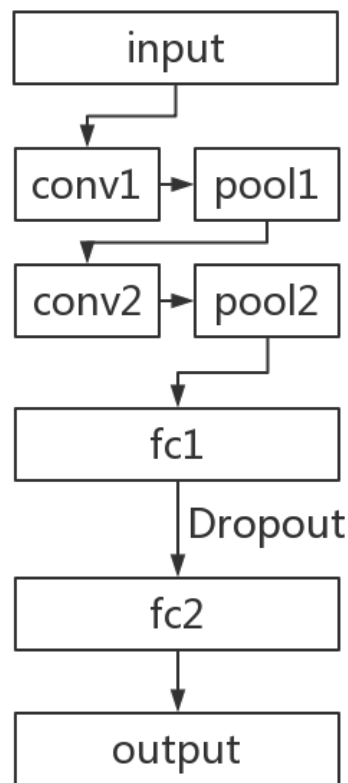


Figure 1: My CNN Architecture

In Python, the code for it is as follows:

```

1  def weight_variable(shape):
2      initial = tf.truncated_normal(shape, stddev=0.1)
3      return tf.Variable(initial)
4
5  def bias_variable(shape):
6      initial = tf.constant(0.1, shape=shape)
7      return tf.Variable(initial)
8
9  # convolution
10 def conv2d(x, W):
11     return tf.nn.conv2d(x, W, strides=[1, 1, 1, 1], padding='VALID')
12
13 # pooling
14 def max_pool_2x2(x):
15     return tf.nn.max_pool(x, ksize=[1, 2, 2, 1], strides=[1, 1, 1, 1],
16         ↪ padding='SAME')
17
18 x = tf.placeholder("float", [None, 9])
19 y_ = tf.placeholder("float", [None, 2])
20
21 # first convolutinal layer
22 w_conv1 = weight_variable([1, 1, 1, 32])
23 b_conv1 = bias_variable([32])
24 x_image = tf.reshape(x, [-1, 3, 3, 1])
25 h_conv1 = tf.nn.relu(conv2d(x_image, w_conv1) + b_conv1)
  
```

```

20 h_pool1 = max_pool_2x2(h_conv1)

21 # second convolutional layer
22 w_conv2 = weight_variable([1, 1, 32, 64])
23 b_conv2 = bias_variable([64])
24 h_conv2 = tf.nn.relu(conv2d(h_pool1, w_conv2) + b_conv2)
25 h_pool2 = max_pool_2x2(h_conv2)

26 # densely connected layer
27 w_fc1 = weight_variable([3*3*64, 1024])
28 b_fc1 = bias_variable([1024])
29 h_pool2_flat = tf.reshape(h_pool2, [-1, 3*3*64])
30 h_fc1 = tf.nn.relu(tf.matmul(h_pool2_flat, w_fc1) + b_fc1)

31 # dropout
32 keep_prob = tf.placeholder("float")
33 h_fc1_drop = tf.nn.dropout(h_fc1, keep_prob)

34 # readout layer
35 w_fc2 = weight_variable([1024, 2])
36 b_fc2 = bias_variable([2])
37 y_conv = tf.nn.softmax(tf.matmul(h_fc1_drop, w_fc2) + b_fc2)

```

Note that when initializing weight matrix and bias vectors, I define functions to initialize them with small random numbers. The purpose is to avoid problems such as "Dead Relu".

5.2.2 Loading Data

When loading data, data need to be preprocessed like normalized. And in accordance with Tensorflow, the label of data should be in the form of one hot vector, like [0,1] and [1,0]. So, I implement a class in load_data.py:

```

1 class data_set(object):
2     def __init__(self):
3         X_train, y_train, X_test, y_test = load_data()
4         self.y_train = np.array(self.one_hot(y_train))
5         self.y_test = np.array(self.one_hot(y_test))
6         self.normalizer = preprocessing.Normalizer().fit(np.array(X_train))
7         self.X_train = self.normalizer.transform(np.array(X_train))
8         self.X_test = self.normalizer.transform(np.array(X_test))

9     def one_hot(self, y):
10        result = list()
11        for yi in y:
12            if yi:
13                result.append([0, 1])
14            else:
15                result.append([1, 0])
16        return result

17    def train_next_batch(self, batch_size):
18        mask = np.random.random_integers(
19            0, self.X_train.shape[0]-1, batch_size)
20        return (self.X_train[mask], self.y_train[mask])

```

5.2.3 Training and Testing

After constructing the CNN, the loss function and optimizer are also needed. For simplicity and efficiency, I choose cross entropy and Adam Optimizer.

```

1 cross_entropy = -tf.reduce_sum(y_*tf.log(y_conv))
2 train_step = tf.train.AdamOptimizer(1e-4).minimize(cross_entropy)

```

Then, begin to train and evaluate:

```

1 sess = tf.InteractiveSession()
2 correct_prediction = tf.equal(tf.argmax(y_conv, 1), tf.argmax(y_, 1))
3 accuracy = tf.reduce_mean(tf.cast(correct_prediction, "float"))
4 sess.run(tf.initialize_all_variables())
5 for i in range(20000):
6     batch = data_set.train_next_batch(128)
7     if i % 100 == 0:
8         train_accuracy = accuracy.eval(
9             feed_dict={x: batch[0], y_: batch[1], keep_prob: 1.0})
10        print("step {0}, train accuracy {1}".format(i, train_accuracy))
11        train_step.run(feed_dict={x: batch[0], y_: batch[1], keep_prob: 0.5})
12 print("test accuracy {}".format(accuracy.eval( feed_dict = {x: data_set.X_test, y_:
    ↪ data_set.y_test, keep_prob: 1.0})))

```

5.2.4 Some Hyper-parameters

As is often the case, training a CNN is effected by quite a few hyper-parameters like learning rate, dropout probability, batch size, etc. To achieve better performance, these hyper-parameters need to be optimized carefully.

For example, as for the learning rate, when I switch it to 1e-2, accuracy changes quickly during training but finally stays at a lower number than usual.

5.2.5 Result

Unfortunately, my computer can't afford too much training and thus I can't adjust all the hyper-parameters to the best. So the result of my CNN is not very distinguishing : only 80%, slightly higher than DNN and SVM.

Considering the huge growth of training time, this result is a little disappointing.

6 Conclusion And Prospect

Doing this project, I have gained a better understanding of and a deeper insight into machine learning and deep learning. Considering the scale of the problem, the advantages of deep learning are not evident enough. In fact, in my opinion, for this problem, traditional machine learning models like SVM works even better than DNN and CNN.

But anyway, I have learned the basic usage of them and I can apply these methods in the future. That's the meaning of this project.