In [1]:

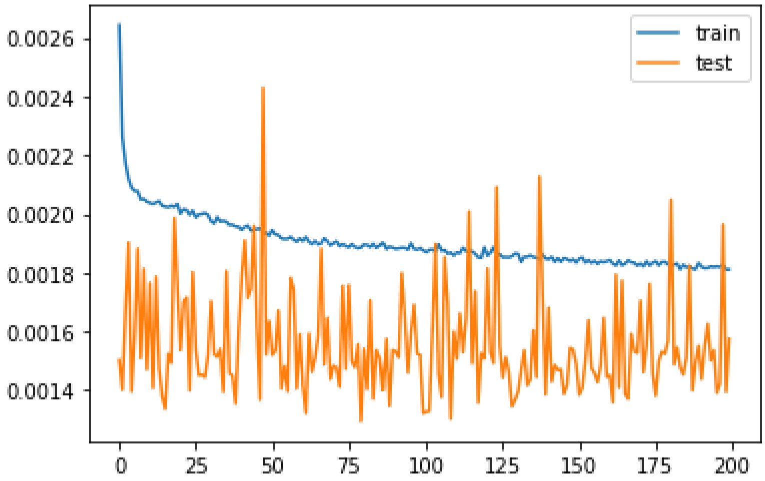
```
from math import sqrt
import tensorflow as tf
import pandas as pd
from keras import Sequential
from keras.layers import LSTM, Dense, Conv1D, GRU
from keras.losses import mean_squared_error
from numpy.core._multiarray_umath import concatenate
from sklearn.preprocessing import MinMaxScaler
import matplotlib.pyplot as plt

# supervised监督学习函数


def series_to_supervised(data, columns, n_in=1, n_out=1, dropnan=True):
    n_vars = 1 if isinstance(data, list) else data.shape[1]
    df = pd.DataFrame(data)
    cols, names = list(), list()
    # input sequence (t-n, ... t-1)
    for i in range(n_in, 0, -1):
        cols.append(df.shift(i))
        names += [('%s%d(t-%d)' % (columns[j], j + 1, i))
                  for j in range(n_vars)]
    # forecast sequence (t, t+1, ... t+n)
    for i in range(0, n_out):
        cols.append(df.shift(-i))
        if i == 0:
            names += [('%s%d(t)' % (columns[j], j + 1)) for j in range(n_vars)]
        else:
            names += [('%s%d(t+%d)' % (columns[j], j + 1, i))
                      for j in range(n_vars)]
    # put it all together
    agg = pd.concat(cols, axis=1)
    agg.columns = names
    # drop rows with NaN values
    if dropnan:
        clean_agg = agg.dropna()
    return clean_agg
    # return agg


dataset = pd.read_csv(
    'Machine_usage_groupby.csv')

dataset_columns = dataset.columns
values = dataset.values
print(dataset)

# 归一化处理
scaler = MinMaxScaler(feature_range=(0, 1))
scaled = scaler.fit_transform(values)


# 监督学习
reframed = series_to_supervised(scaled, dataset_columns, 1, 1)
values = reframed.values

# 学习与检测数据的划分
n_train_hours = 20000
train = values[:n_train_hours, :]
```

```
test = values[n_train_hours:, :]


# 监督学习结果划分
train_x, train_y = train[:, :-1], train[:, -1]
test_x, test_y = test[:, :-1], test[:, -1]

#reshape
train_X = train_x.reshape((train_x.shape[0], 1, train_x.shape[1]))
test_X = test_x.reshape((test_x.shape[0], 1, test_x.shape[1]))


model = Sequential()
model.add(Conv1D(filters=64, kernel_size=10,
                 strides=1, padding="causal",
                 activation="sigmoid"))
model.add(
    GRU(
        64,
        input_shape=(
            train_X.shape[1],
            train_X.shape[2]),
        return_sequences=True))
model.add(GRU(32, input_shape=(train_X.shape[1], train_X.shape[2])))
model.add(Dense(32, activation="relu"))
model.add(Dense(16, activation="relu"))
model.add(Dense(1))
model.compile(loss=tf.keras.losses.Huber(),
    optimizer='adam',
    metrics=["mae"])
history = model.fit(
    train_X,
    train_y,
    epochs=200,
    batch_size=72,
    validation_data=(
        test_X,
        test_y),
verbose = 0)
```

```
Using TensorFlow backend.

         Time  cpu_util_percent  ...  net_out  disk_usage_percent
0           0              12.0  ...    26.49                31.0
1         130              13.5  ...    28.63                 2.5
2         140               5.0  ...    18.12                 1.0
3         160              13.0  ...    30.09                 3.0
4         180               5.0  ...    18.12                 1.0
...       ...               ...  ...      ...                 ...
36211  691040              28.0  ...    41.75                 3.0
36212  691060              93.0  ...    38.37                 3.0
36213  691080              31.0  ...    35.73                 3.0
36214  691110              92.0  ...    38.37                 2.0
36215  691180              18.5  ...    38.47                 3.5

[36216 rows x 8 columns]
```

In [2]:

```
#画图
plt.plot(history.history['loss'], label='train')
plt.plot(history.history['val_loss'], label='test')
plt.legend()
plt.show()
```



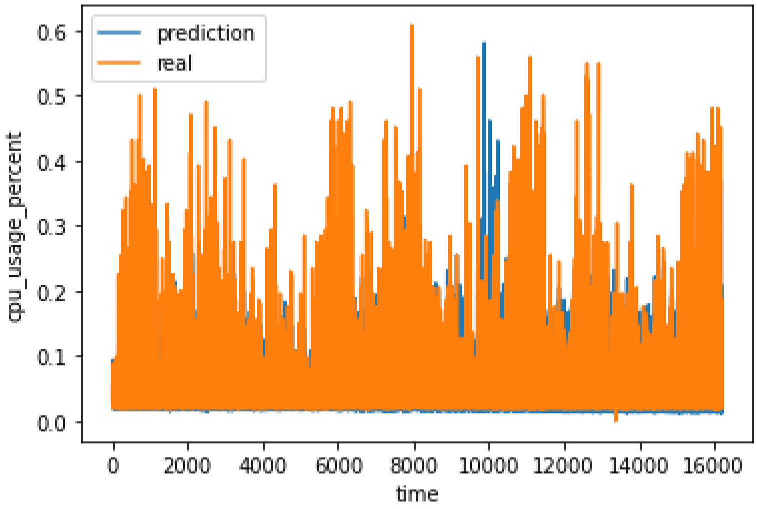
In [3]:

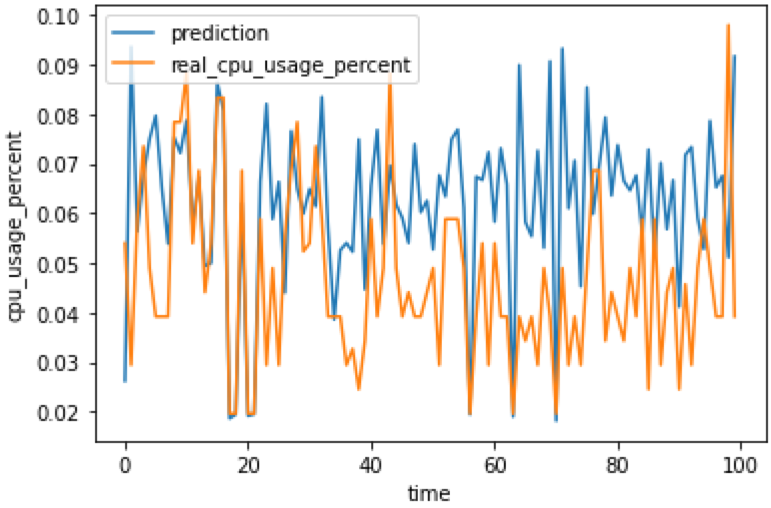
```
# make the prediction
yHat = model.predict(test_X)

inv_yHat = concatenate((yHat, test_x[:, 1:]), axis=1)   # 数组拼接
inv_yHat = inv_yHat[:, 0]

test_y = test_y.reshape((len(test_y), 1))
inv_y = concatenate((test_y, test_x[:, 1:]), axis=1)
inv_y = inv_y[:, 0]

rmse = sqrt(mean_squared_error(inv_yHat, inv_y))
print('Test RMSE: %.3f' % rmse)
```

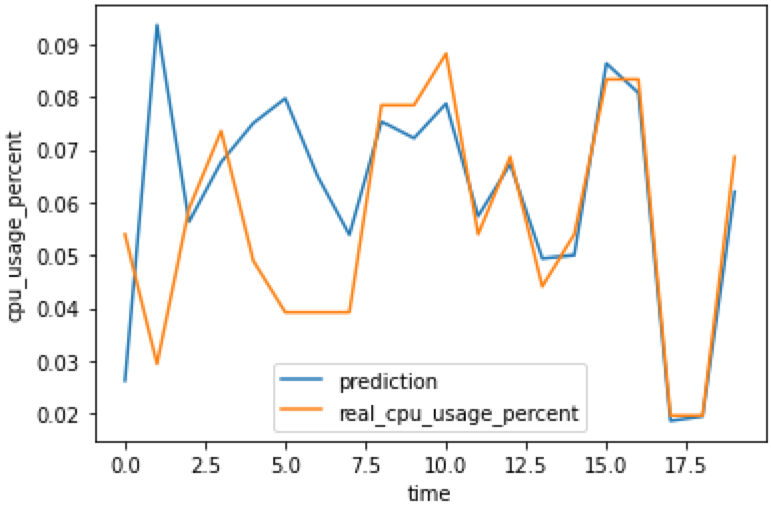
```
Test RMSE: 0.056
```

In [4]:

```
yhat = model.predict(test_X)
test_X_reshaped = test_X.reshape((test_X.shape[0], test_X.shape[2]))

inv_yhat = concatenate((yhat, yhat, test_X_reshaped[:, 1:]), axis=1)
inv_yhat = inv_yhat[:, 0]
test_y = test_y.reshape((len(test_y), 1))
inv_y = concatenate((test_y, test_y, test_X_reshaped[:, 1:]), axis=1)
inv_y = inv_y[:, 0]
plt.plot(inv_yhat, label='prediction')
plt.plot(inv_y, label='real')
plt.xlabel('time')
plt.ylabel('cpu_usage_percent')
plt.legend()
plt.show()
plt.plot(inv_yhat[:100], label='prediction')
plt.plot(inv_y[:100], label='real_cpu_usage_percent')
plt.xlabel('time')
plt.ylabel('cpu_usage_percent')
plt.legend()
plt.show()
plt.plot(inv_yhat[:20], label='prediction')
plt.plot(inv_y[:20], label='real_cpu_usage_percent')
plt.xlabel('time')
plt.ylabel('cpu_usage_percent')
plt.legend()
plt.show()
```




prediction
real_cpu_usage_percent
cpu_usage_percent
0.10
0.09
0.08
0.07
0.06
0.05
0.04
0.03
0.02
0
20
40
60
80
100
time


prediction
real_cpu_usage_percent
cpu_usage_percent
0.09
0.08
0.07
0.06
0.05
0.04
0.03
0.02
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
time