

Lab 3: Indoor Localization Based on Wi-Fi Fingerprinting and Deep Neural Networks

SCHOOL OF ADVANCED TECHNOLOGY

CAN405: Data Communication and Communications Networks

Mengfan Li

ID number: 2032048

2020 Fall

GitHub link: <https://github.com/Mengfan-Li/Indoor-location>

Content

1	Objectives.....	1
2	Materials.....	1
3	Methods.....	1
3.1	UJIIndoorLoc dataset	1
3.1.1	Data introduction.....	1
3.1.2	load dataset.....	3
3.2	Indoor Location Algorithm.....	4
3.2.1	TensorFlow and Keras	4
3.2.2	Indoor Location Algorithm kernel	4
3.2.3	SAE.....	5
3.2.4	ont-hot	7
3.2.5	DNN	8
3.2.6	Dropout	10
4	Lab Task.....	11
4.2	Implementation results.	11
4.3	Analysis and discussions.	20
5	Conclusion	33

1 Objectives

In this Lab, which is designed open-ended, you are to build a prototype of indoor localization system based on Wi-Fi fingerprinting and deep neural networks (DNNs) and demonstrate its localization performance with the UJIIndoorLoc multi-building and multi-floor dataset.

2 Materials

Pycharm

3 Methods

3.1 UJIIndoorLoc dataset

3.1.1 Data introduction

The UJIIndoorLoc database[1] covers three buildings of Universitat Jaume I (<http://www.uji.es>) with 4 or more floors and almost 110.000m². It can be used for classification, e.g. actual building and floor identification, or regression, e.g. actual longitude and latitude estimation. It was created in 2013 by means of more than 20 different users and 25 Android devices. The database consists of 19937 training/reference records (trainingData.csv file) and 1111 validation/test records (validationData.csv file)

The 529 attributes contain the WiFi fingerprint, the coordinates where it was taken, and other useful information.

Each WiFi fingerprint can be characterized by the detected Wireless Access Points (WAPs) and the corresponding Received Signal Strength Intensity (RSSI). The intensity values are represented as negative integer values ranging -104dBm (extremely poor signal) to 0dbM. The positive value 100 is used to denote when a WAP was not detected. During the database creation, 520 different WAPs were detected. Thus, the WiFi fingerprint is composed by 520 intensity values.

Then the coordinates (latitude, longitude, floor) and Building ID are provided as the attributes

to be predicted.

The particular space (offices, labs, etc.) and the relative position (inside/outside the space) where the capture was taken have been recorded. Outside means that the capture was taken in front of the door of the space.

Information about who (user), how (android device & version) and when (timestamp) WiFi capture was taken is also recorded.

- Attributes 001 to 520 (WAP001-WAP520): Intensity value for WAP001. Negative integer values from -104 to 0 and +100. Positive value 100 used if WAP001 was not detected.
- Attribute 521 (Longitude): Longitude. Negative real values from -7695.9387549299299000 to -7299.786516730871000
- Attribute 522 (Latitude): Latitude. Positive real values from 4864745.7450159714 to 4865017.3646842018.
- Attribute 523 (Floor): Altitude in floors inside the building. Integer values from 0 to 4.
- Attribute 524 (BuildingID): ID to identify the building. Measures were taken in three different buildings. Categorical integer values from 0 to 2.
- Attribute 525 (SpaceID): Internal ID number to identify the Space (office, corridor, classroom) where the capture was taken. Categorical integer values.
- Attribute 526 (RelativePosition): Relative position with respect to the Space (1 - Inside, 2 - Outside in Front of the door). Categorical integer values.
- Attribute 527 (UserID): User identifier (see below). Categorical integer values.
- Attribute 528 (PhoneID): Android device identifier (see below). Categorical integer values.
- Attribute 529 (Timestamp): UNIX Time when the capture was taken. Integer value.

	SY	SZ	TA	TB	TC	TD	TE	TF	TG	TH	TI	T
.8	WAP519	WAP520	LONGITUDE	LATITUDE	FLOOR	BUILDINGID	SPACEID	RELATIVEPOSITION	USERID	PHONEID	TIMESTAMP	
110	-110	-110	-7515.9168	4864889.663	1	1	0	0	0	0	1.38E+09	
110	-110	-110	-7383.86722	4864839.74	4	2	0	0	0	13	1.38E+09	
110	-110	-110	-7374.30208	4864846.534	4	2	0	0	0	13	1.38E+09	
110	-110	-110	-7365.82488	4864842.829	4	2	0	0	0	13	1.38E+09	
110	-110	-110	-7641.4993	4864922.399	2	0	0	0	0	2	1.38E+09	
110	-110	-110	-7338.80731	4864835.467	2	2	0	0	0	13	1.38E+09	

3.1.2 load dataset

Code:

```
1. path_train = '../data/UJIIndoorLoc/trainingData2.csv' # '-
110' for the lack of AP.
2. path_validation = '../data/UJIIndoorLoc/validationData2.csv' # ditto
3.
4. train_df = pd.read_csv(path_train, header=0) # pass header=0 to be able to replace
existing names
5. test_df = pd.read_csv(path_validation, header=0)
```

Result:

```
176 # read both train and test dataframes for consistent label formation through one-hot encoding
177 train_df = pd.read_csv(path_train, header=0) # pass header=0 to be able to replace existing names
178 test_df = pd.read_csv(path_validation, header=0)
179 print("train data")
180 print(train_df)
181
if __name__ == "__main__":
```

运行: scalable_indoor_localization x

	WAP001	WAP002	WAP003	...	USERID	PHONEID	TIMESTAMP
0	-110	-110	-110	...	2	23	1371713733
1	-110	-110	-110	...	2	23	1371713691
2	-110	-110	-110	...	2	23	1371714095
3	-110	-110	-110	...	2	23	1371713807
4	-110	-110	-110	...	11	13	1369909710
...
19932	-110	-110	-110	...	18	10	1371710683
19933	-110	-110	-110	...	18	10	1371710402
19934	-110	-110	-110	...	18	10	1371710921
19935	-110	-110	-110	...	18	10	1371711049
19936	-110	-110	-110	...	18	10	1371711025

[19937 rows x 529 columns]

```
176 # read both train and test dataframes for consistent label formation through one-hot encoding
177 train_df = pd.read_csv(path_train, header=0) # pass header=0 to be able to replace existing names
178 test_df = pd.read_csv(path_validation, header=0)
179 print("test data")
180 print(test_df)
181
if __name__ == "__main__":
```

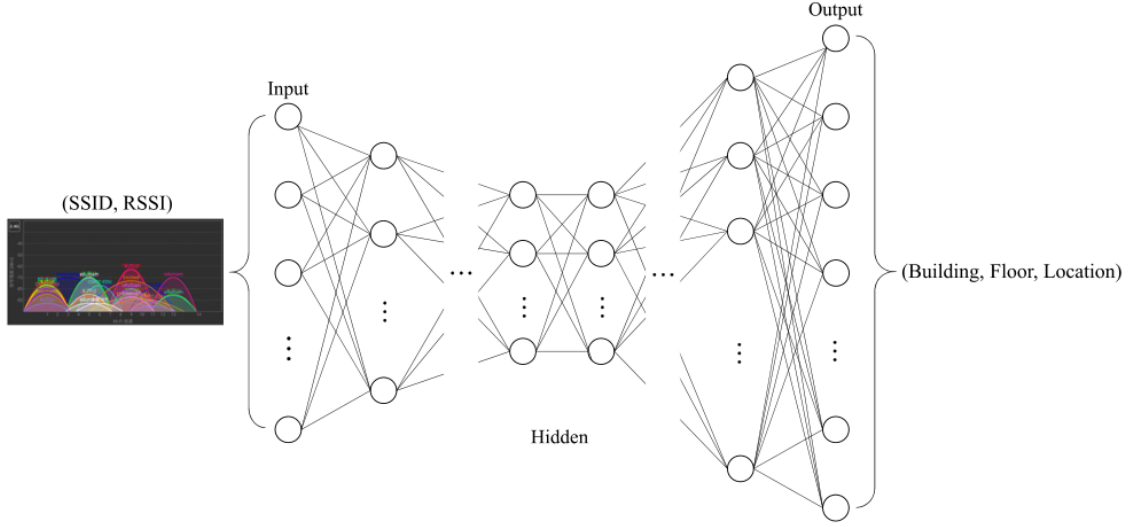
运行: scalable_indoor_localization x

I:\Anaconda\python.exe C:/Users/123/Desktop/can405_indoor_localization-master/python/scalable_indoor_localization.py

	WAP001	WAP002	WAP003	...	USERID	PHONEID	TIMESTAMP
0	-110	-110	-110	...	0	0	1380872703
1	-110	-110	-110	...	0	13	1381155054
2	-110	-110	-110	...	0	13	1381155095
3	-110	-110	-110	...	0	13	1381155138
4	-110	-110	-110	...	0	2	1380877774
...
1106	-110	-110	-110	...	0	13	1381156711
1107	-110	-110	-110	...	0	13	1381156730
1108	-110	-110	-110	...	0	13	1381247781
1109	-110	-110	-110	...	0	13	1381247807
1110	-110	-110	-110	...	0	13	1381247836

[1111 rows x 529 columns]

3.2 Indoor Location Algorithm



Overall network structure diagram[2]

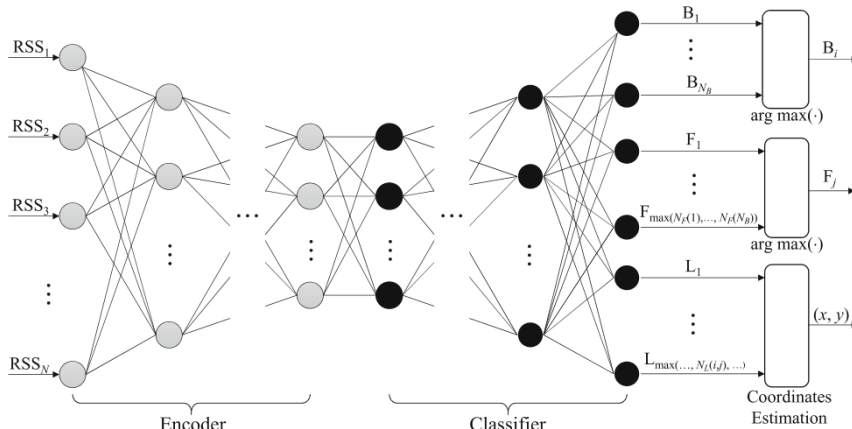
3.2.1 TensorFlow and Keras

In this experiment, the main TensorFlow framework used, the Keras network library. For more information, please check the official manual:

- <https://tensorflow.google.cn/>
- <https://keras.io/>

3.2.2 Indoor Location Algorithm kernel

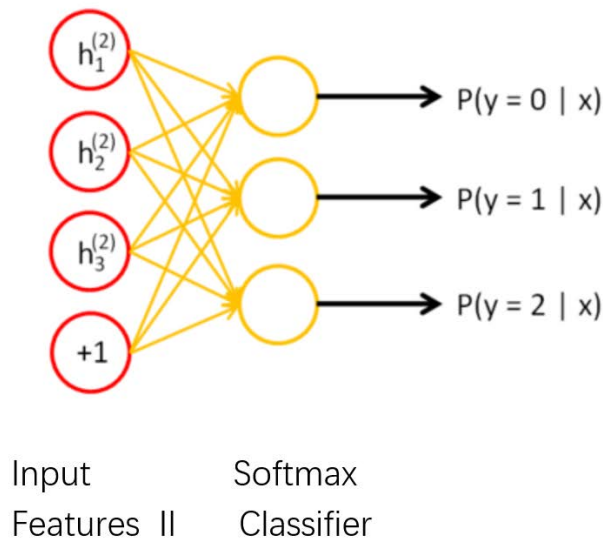
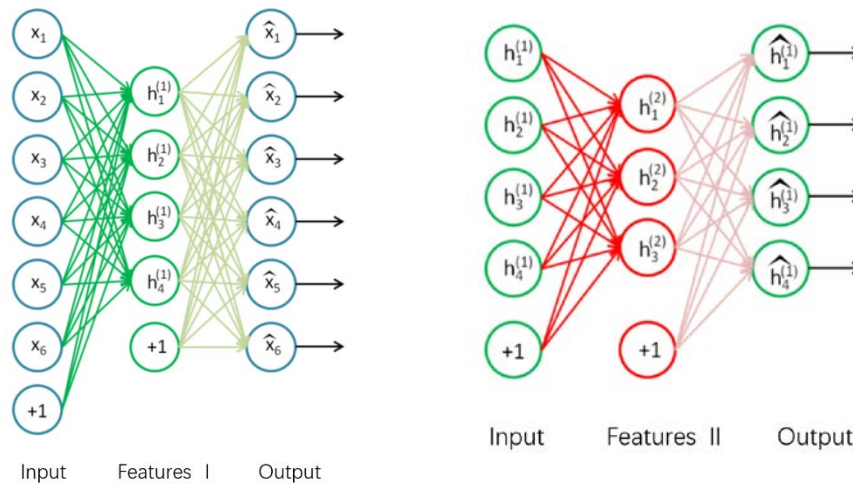
To address the scalability issue of the DNN classifier based on multi-class classification and take into account the hierarchical nature of the building/floor/location classification, we propose a scalable DNN architecture based on multi-label classification.

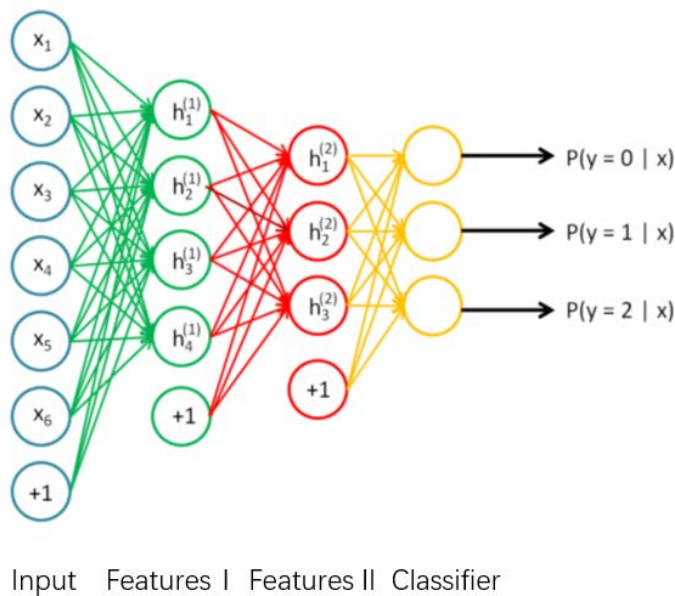


3.2.3 SAE

As the name suggests, a stacked autoencoder is a cascade of multiple autoencoders to complete the task of layer-by-layer feature extraction. The final features are more representative and have small dimensions.[3]

The training process of stacked autoencoder is that n AEs are trained in sequence. After the first AE is trained, the output of its encoder is used as the input of the second AE, and so on. The finally obtained features are used as the input of the classifier to complete the final classification training. As shown in the following four pictures:





After training layer by layer, a fine tuning process (Fine tuning) is needed. The general idea is as follows: After layer-by-layer training, the weight of each layer of AE and the weight of the softmax classification layer already have a pretrain value. At this time, we will connect the entire network and use the data for a training session, so that the weight parameters of each layer are at the same time Improved.

Code:

```

1. # SAE_ACTIVATION = 'tanh'
2. SAE_ACTIVATION = 'relu'
3. SAE_BIAS = False
4. SAE_OPTIMIZER = 'adam'
5. SAE_LOSS = 'mse'

```

It should be noted here that in the gradient descent update, the Adam optimization algorithm is used.

Adam has the following advantages:

- High computational efficiency
- Little memory requirement
- The diagonal rescaling of the gradient is unchanged (this means that Adam multiplying the gradient by the diagonal matrix with only positive factors is unchanged in order to better

understand this stack exchange)

- Ideal for problems with large data and/or parameters
- Suitable for non-fixed targets
- Suitable for problems with very noisy and/or sparse gradients
- Hyperparameters have intuitive explanations and usually require very little adjustment (we will cover this in detail in the configuration section)

In short, Adam uses momentum and adaptive learning rate to speed up convergence.

```
Require:  $\alpha$ : Stepsize  
Require:  $\beta_1, \beta_2 \in [0, 1)$ : Exponential decay rates for the moment estimates  
Require:  $f(\theta)$ : Stochastic objective function with parameters  $\theta$   
Require:  $\theta_0$ : Initial parameter vector  
 $m_0 \leftarrow 0$  (Initialize 1st moment vector)  
 $v_0 \leftarrow 0$  (Initialize 2nd moment vector)  
 $t \leftarrow 0$  (Initialize timestep)  
while  $\theta_t$  not converged do  
   $t \leftarrow t + 1$   
   $g_t \leftarrow \nabla_{\theta} f_t(\theta_{t-1})$  (Get gradients w.r.t. stochastic objective at timestep  $t$ )  
   $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$  (Update biased first moment estimate)  
   $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$  (Update biased second raw moment estimate)  
   $\hat{m}_t \leftarrow m_t / (1 - \beta_1^t)$  (Compute bias-corrected first moment estimate)  
   $\hat{v}_t \leftarrow v_t / (1 - \beta_2^t)$  (Compute bias-corrected second raw moment estimate)  
   $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \hat{m}_t / (\sqrt{\hat{v}_t} + \epsilon)$  (Update parameters)  
end while  
return  $\theta_t$  (Resulting parameters)
```

In the end, using the SAE network, the data scale was halved.

Code:

```
1.  # remove the decoder part  
2.  num_to_remove = (len(sae_hidden_layers) + 1) // 2
```

3.2.4 ont-hot

The basic idea of one-hot: treat each value of a discrete feature as a state. If your feature has N different values, then we can abstract the feature into N types. For different states, the one-hot encoding ensures that each value will only make one state in the "active state", that is to say, only one state bit of the N states has a value of 1, and the other state bits are 0.[4]

Code:

```
1. blds_all = np.asarray(pd.get_dummies(pd.concat([train_df['BUILDINGID'], test_df['BUILDINGID']])) # for consistency in one-hot encoding for both dataframes
2. flrs_all = np.asarray(pd.get_dummies(pd.concat([train_df['FLOOR'], test_df['FLOOR']])) # ditto
```

Result:

```
~~~~~one-hot~~~~~
0      1
1      1
2      1
3      1
4      0
..
19932   1
19933   2
19934   1
19935   1
19936   1
Name: BUILDINGID, Length: 19937, dtype: int64
~~~~~after one-hot~~~~~
[[0 1 0]
 [0 1 0]
 [0 1 0]
 ...
 [1 0 0]
 [1 0 0]
 [1 0 0]]
```

3.2.5 DNN

Neural network is based on the extension of the perceptron, and DNN can be understood as a neural network with many hidden layers. Multi-layer neural network and deep neural network DNN actually refer to one thing. DNN is sometimes called Multi-Layer perceptron (MLP).[5]

According to the location of different layers of DNN, the neural network layer inside DNN can be divided into three categories, input layer, hidden layer and output layer, as shown in the example below. Generally speaking, the first layer is the input layer and the last layer is the output layer. And the number of layers in the middle are all hidden layers.

The layers are fully connected, that is, any neuron in the i -th layer must be connected to any neuron in the $i+1$ th layer. Although DNN looks very complicated, but from the perspective of a small

local model, it is still the same as the perceptron, that is, a linear relationship $z = \sum w_i x_i + b$, and an activate function $\sigma(z)$.

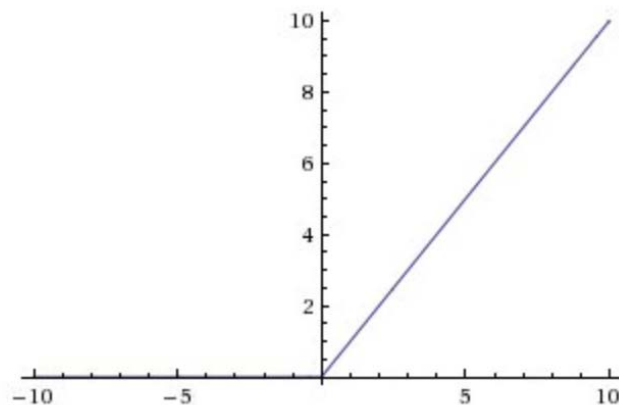
Code:

```
1. CLASSIFIER_ACTIVATION = 'relu'
2. CLASSIFIER_BIAS = False
3. CLASSIFIER_OPTIMIZER = 'adam' # learning rate
4. CLASSIFIER_LOSS = 'binary_crossentropy'
```

- Relu:

The Rectified Linear Unit is the most commonly used activation function in deep learning models. The function returns 0 if it receives any negative input, but for any positive value x it returns that value back. So it can be written as $f(x) = \max(0, x)$.

Graphically it looks like this



- binary_crossentropy

The algorithm is to select the average value of a small part of the sample gradient to update the weight (where $n < m$, m is the number of samples)

$$w_j = w_j - \frac{\alpha}{n} \sum_{i=1}^n (y_i - \pi(x_i)) x_{i,j}$$

Build model Code:

```
1. model.add(Dropout(dropout))
2. for i in range(len(classifier_hidden_layers)):
3.     model.add(Dense(classifier_hidden_layers[i], name="classifier-hidden"+str(i),
4.                     activation=CLASSIFIER_ACTIVATION, use_bias=CLASSIFIER_BIAS))
```

```

5.         model.add(Dropout(dropout))
6.     model.add(Dense(OUTPUT_DIM, name="activation-0", activation='sigmoid',
7.                     use_bias=CLASSIFIER_BIAS)) # 'sigmoid' for multi-
label classification
8.     model.compile(optimizer=CLASSIFIER_OPTIMIZER, loss=CLASSIFIER_LOSS, metrics=['accuracy'])

```

3.2.6 Dropout

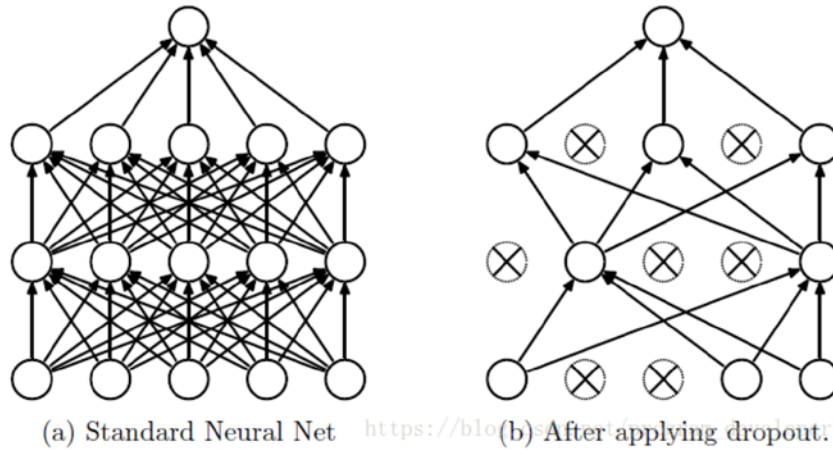
In a machine learning model, if the model has too many parameters and too few training samples, the trained model is prone to overfitting. The problem of over-fitting is often encountered when training neural networks. Over-fitting is specifically manifested in: the model has a smaller loss function on the training data and higher prediction accuracy; but on the test data, the loss function is relatively large, and the prediction. The accuracy rate is low.[6]

Overfitting is a common problem in many machine learning. If the model is overfitted, the resulting model is almost unusable. In order to solve the problem of overfitting, the method of model integration is generally adopted, that is, training multiple models to combine. At this time, the time-consuming training model becomes a big problem. Not only is it time-consuming to train multiple models, but it is also time-consuming to test multiple models.

Dropout can effectively alleviate the occurrence of over-fitting, and achieve the effect of regularization to a certain extent.

Dropout can be used as a trick for training deep neural networks. In each training batch, by ignoring half of the feature detectors (making half of the hidden layer nodes value 0), over-fitting can be significantly reduced. This method can reduce the interaction between feature detectors (hidden layer nodes). Detector interaction means that some detectors rely on other detectors to function.

The simple point of Dropout is: when we propagate forward, let the activation value of a certain neuron stop working with a certain probability p , which can make the model more generalized, because it will not rely too much on certain parts The characteristics are as shown.



4 Lab Task

4.2 Implementation results.

- Proper working of the implemented algorithm(s) without errors during runtime.

Because my laptop does not have TensorFlow installed, there will be a lot of error messages about TF when it is running.

```
scalable_indoor_localization (1)
I:\Anaconda\python.exe "C:/Users/123/Desktop/can405_indoor_localization-master (2)
/can405_indoor_localization-master/python/scalable_indoor_localization.py"
2021-01-23 18:24:19.623427: I tensorflow/stream_executor/platform/default/dso_loader.cc:49] Successfully opened dynamic library cudart64_110.dll

Part 1: buidling an SAE encoder ...
2021-01-23 18:24:23.236358: I tensorflow/compiler/jit/xla_cpu_device.cc:41] Not creating XLA devices, tf_xla_enable_xla_devices not set
2021-01-23 18:24:23.238691: I tensorflow/stream_executor/platform/default/dso_loader.cc:49] Successfully opened dynamic library nvcuda.dll
2021-01-23 18:24:24.094929: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1720] Found device 0 with properties:
pciBusID: 0000:02:00.0 name: GeForce MX350 computeCapability: 6.1
coreClock: 1.468GHz coreCount: 5 deviceMemorySize: 2.00GiB deviceMemoryBandwidth: 52.21GiB/s
2021-01-23 18:24:24.095278: I tensorflow/stream_executor/platform/default/dso_loader.cc:49] Successfully opened dynamic library cudart64_110.dll
2021-01-23 18:24:24.096128: W tensorflow/stream_executor/platform/default/dso_loader.cc:68] Could not load dynamic library 'cublas64_11.dll';
dlerror: cublas64_11.dll not found
2021-01-23 18:24:24.096884: W tensorflow/stream_executor/platform/default/dso_loader.cc:68] Could not load dynamic library 'cublasLt64_11.dll';
dlerror: cublasLt64_11.dll not found
2021-01-23 18:24:24.100247: I tensorflow/stream_executor/platform/default/dso_loader.cc:49] Successfully opened dynamic library cufft64_10.dll
2021-01-23 18:24:24.101638: I tensorflow/stream_executor/platform/default/dso_loader.cc:49] Successfully opened dynamic library curand64_10.dll
2021-01-23 18:24:24.105240: I tensorflow/stream_executor/platform/default/dso_loader.cc:49] Successfully opened dynamic library cusolver64_10
.dll
2021-01-23 18:24:24.106059: W tensorflow/stream_executor/platform/default/dso_loader.cc:68] Could not load dynamic library 'cusparse64_11.dll';
dlerror: cusparse64_11.dll not found
2021-01-23 18:24:24.106879: W tensorflow/stream_executor/platform/default/dso_loader.cc:68] Could not load dynamic library 'cudnn64_8.dll';
dlerror: cudnn64_8.dll not found
```

You can see that there will be a lot of red warning messages. These are reminders that there is no TF environment. Therefore, in order to make the program more beautiful and not affect the observation results, we add a line of code to block the TensorFlow information reminder.

```
1. os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'
```

Result:

```
scalable_indoor_localization (1) x
I:\Anaconda\python.exe "C:/Users/123/Desktop/can405_indoor_localization-master (2)
/can405_indoor_localization-master/python/scalable_indoor_localization.py"

Part 1: buidling an SAE encoder ...
```

Step1: load dataset

```
#-----load dataset successful-----
path_train = '../data/UJIIndoorLoc/trainingData2.csv' # '-110' for the lack of AP.
path_validation = '../data/UJIIndoorLoc/validationData2.csv' # ditto
print("-----load dataset successful-----")

scalable_indoor_localization (1) x
I:\Anaconda\python.exe "C:/Users/123/Desktop/can405_indoor_localization-master (2)
/can405_indoor_localization-master/python/scalable_indoor_localization.py"
-----load dataset successful-----
```

Step2: Train and validation data

```
1. # split the training set into training and validation sets; we will use the
2. # validation set at a testing set.
3. train_val_split = np.random.rand(len(train_AP_features)) < training_ratio # mask ind
ex array
4. x_train = train_AP_features[train_val_split]
5. y_train = train_labels[train_val_split]
6. x_val = train_AP_features[~train_val_split]
7. y_val = train_labels[~train_val_split]
```

Result:

```
> x_avg = {dict: 15} {'0-0': -7640.376350991143, '0-1': -7637.829042353956, '0-2': -7639.023285886209, '0-3': -7640.245117900643, '0-4': nan, '1-0': -7461... (显示)
> x_train = (ndarray: (17928, 520)) [[-0.17125017 -0.17125017 -0.17125017 ... -0.17125017 -0.17125017, -0.17125017, -0.16059846 -0.16059846...View as Array
> x_val = (ndarray: (2009, 520)) [[-0.17119193 -0.17119193 -0.17119193 ... -0.17119193 -0.17119193, -0.17119193, [-0.17455084 -0.17455084 -C...View as Array
> y_avg = {dict: 15} {'0-0': 4864957.72195798, '0-1': 4864957.706202947, '0-2': 4864958.3644638825, '0-3': 4864956.533051689, '0-4': nan, '1-0': 4864874 ... (显示)
> y_train = (ndarray: (17928, 118)) [[0 1 0 ... 0 0 0], [0 1 0 ... 0 0 0], [0 1 0 ... 0 0 0], ..., [0 0 1 ... 0 0 0], [0 1 0 ... 0 0 0], [0 0 1 ... 0 0 0]] ...View as Array
> y_val = (ndarray: (2009, 118)) [[0 1 0 ... 0 0 0], [0 1 0 ... 0 0 0], [0 1 0 ... 0 0 0], ..., [0 1 0 ... 0 0 0], [0 1 0 ... 0 0 0], [0 1 0 ... 0 0 0]] ...View as Array
```


Due to the one-hot operation on the label, it looks like a list.

	÷ 0	÷ 1	÷ 2	÷ 3	÷ 4	÷ 5	÷ 6	÷ 7	÷ 8	÷ 9	÷ 10
0	0	1	0	0	0	1	0	0	0	0	0
1	0	1	0	0	0	1	0	0	0	0	0
2	0	1	0	0	0	1	0	0	0	0	0
3	0	1	0	0	0	1	0	0	0	0	0
4	1	0	0	1	0	0	0	0	0	0	0
5	0	1	0	0	0	1	0	0	0	0	0
6	0	1	0	0	0	1	0	0	0	0	0
7	0	1	0	0	0	1	0	0	0	0	1
8	0	1	0	0	0	1	0	0	0	0	0
9	0	1	0	0	0	1	0	0	0	0	0
10	0	1	0	0	0	1	0	0	0	0	0
11	0	1	0	0	0	1	0	0	0	0	0
12	0	1	0	0	0	1	0	0	0	0	0
13	0	1	0	0	0	1	0	0	0	0	0

Step3: SAE model

Code:

```

1. # create a model based on stacked autoencoder (SAE)
2. model = Sequential()
3. model.add(Dense(sae_hidden_layers[0], name="sae-hidden-
0", input_dim=INPUT_DIM, activation=SAE_ACTIVATION, use_bias=SAE_BIAS))
4. for i in range(1, len(sae_hidden_layers)):
5.     model.add(Dense(sae_hidden_layers[i], name="sae-hidden-
"+str(i), activation=SAE_ACTIVATION, use_bias=SAE_BIAS))
6. model.add(Dense(INPUT_DIM, name="sae-
output", activation=SAE_ACTIVATION, use_bias=SAE_BIAS))
7. model.compile(optimizer=SAE_OPTIMIZER, loss=SAE_LOSS)
8.
9. # train the model
10. model.fit(x_train, x_train, batch_size=batch_size, epochs=epochs, verbose=VERBOSE)
11.
12. # remove the decoder part
13. num_to_remove = (len(sae_hidden_layers) + 1) // 2
14. for i in range(num_to_remove):
15.     model.pop()
16.
17. # set all layers (i.e., SAE encoder) to non-
trainable (weights will not be updated)
18. # for layer in model.layers[:]:
19. #     layer.trainable = False
20.
21. # save the model for later use

```

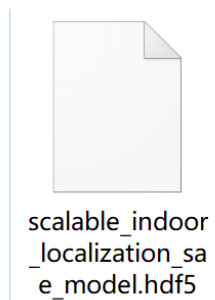
```
22. model.save(path_sae_model)
```

Result:

```
I:\Anaconda\python.exe C:/Users/123/Desktop/can405_indoor_localization-master/python/scalable_indoor_localization.py

Part 1: buidling an SAE encoder ...
Epoch 1/50
180/180 [=====] - 1s 2ms/step - loss: 0.6568
Epoch 2/50
180/180 [=====] - 0s 2ms/step - loss: 0.3559
Epoch 3/50
180/180 [=====] - 0s 2ms/step - loss: 0.3325
Epoch 4/50
180/180 [=====] - 0s 2ms/step - loss: 0.3183
Epoch 5/50
149/180 [=====>.....] - ETA: 0s - loss: 0.3135
```

Save model:



Step4: train and validation model

```
1. # train the model
2. startTime = timer()
3. model.fit(x_train, y_train, validation_data=(x_val, y_val), batch_size=batch_size, epochs=epochs, verbose=VERBOSE)
4. elapsedTime = timer() - startTime
5. print("Model trained in %e s." % elapsedTime)
```

```
Part 2: buidling a complete model ...
Epoch 1/50
180/180 [=====] - 2s 9ms/step - loss: 0.1570 - accuracy: 0.6250 - val_loss: 0.0483 - val_accuracy: 0.9039
Epoch 2/50
180/180 [=====] - 1s 6ms/step - loss: 0.0527 - accuracy: 0.7794 - val_loss: 0.0422 - val_accuracy: 0.7531
Epoch 3/50
180/180 [=====] - 1s 6ms/step - loss: 0.0455 - accuracy: 0.7145 - val_loss: 0.0379 - val_accuracy: 0.7591
Epoch 4/50
180/180 [=====] - 1s 5ms/step - loss: 0.0413 - accuracy: 0.7055 - val_loss: 0.0354 - val_accuracy: 0.7850
Epoch 5/50
180/180 [=====] - 1s 5ms/step - loss: 0.0391 - accuracy: 0.6923 - val_loss: 0.0332 - val_accuracy: 0.7775
Epoch 6/50
180/180 [=====] - 1s 5ms/step - loss: 0.0363 - accuracy: 0.6921 - val_loss: 0.0317 - val_accuracy: 0.7307
```


Result:

```
1793/1793 [=====] - 2s 1ms/step - loss: 0.0118 - accuracy: 0.6907 - val_loss: 0.0206 - val_accuracy: 0.6725
Epoch 19/20
1793/1793 [=====] - 2s 1ms/step - loss: 0.0113 - accuracy: 0.6627 - val_loss: 0.0212 - val_accuracy: 0.7023
Epoch 20/20
1793/1793 [=====] - 2s 1ms/step - loss: 0.0111 - accuracy: 0.6688 - val_loss: 0.0201 - val_accuracy: 0.6725
Model trained in 4.450118e+01 s.
```

Step5: test model

Test data

- ```
1. # turn the given validation set into a testing set
2. test_AP_features = scale(np.asarray(test_df.iloc[:,0:520]).astype(float), axis=1)
3. x_test_utm = np.asarray(test_df['LONGITUDE'])
4. y_test_utm = np.asarray(test_df['LATITUDE'])
5. blds = blds_all[len_train:]
6. flrs = flrs_all[len_train:]
```

```

print("Model trained in %s." % elapsedTime)

turn the given validation set into a testing set
test_AP_features = scale(np.asarray(test_df.iloc[:,0:520])).as
x_test_utm = np.asarray(test_df['LONGITUDE']) x_test_utm:
y_test_utm = np.asarray(test_df['LATITUDE']) y_test_utm:
blds = blds_all[len_train:]
flrs = flrs_all[len_train:]
K = 1

```

|   | ÷ 0      | ÷ 1      | ÷ 2      | ÷ 3      | ÷ 4      | ÷ 5      | ÷ 6      | ÷ 7      |
|---|----------|----------|----------|----------|----------|----------|----------|----------|
| 0 | -0.04390 | -0.04390 | -0.04390 | -0.04390 | -0.04390 | -0.04390 | -0.04390 | -0.04390 |
| 1 | -0.13601 | -0.13601 | -0.13601 | -0.13601 | -0.13601 | -0.13601 | -0.13601 | -0.13601 |
| 2 | -0.12483 | -0.12483 | -0.12483 | -0.12483 | -0.12483 | -0.12483 | -0.12483 | -0.12483 |
| 3 | -0.13191 | -0.13191 | -0.13191 | -0.13191 | -0.13191 | -0.13191 | -0.13191 | -0.13191 |
| 4 | -0.16213 | -0.16213 | -0.16213 | -0.16213 | -0.16213 | -0.16213 | -0.16213 | -0.16213 |
| 5 | -0.21101 | -0.21101 | -0.21101 | -0.21101 | -0.21101 | -0.21101 | -0.21101 | -0.21101 |

```

test_AP_features
格式: %s

```

```

if __name__ == '__main__':
 scalable_indoor_localization(1)

```

变量

```

> rfps = ndarray: (19937, 110) [[0 0 ... 0 0], [0 0 ... 0 0], [0 0 ... 0 0], ..., [0 0 ... 0 0], [0 0 ... 0 0]] View as Array
> sae_hidden_layers = (list: 5) [256, 128, 64, 128, 256]
> scaling = (float) 0.0
> startTime = (float) 68.5259029
> test_AP_features = ndarray: (1111, 520) [-0.04389513 -0.04389513 -0.04389513 ... -0.04389513 -0.04389513, [-0.04389513], [-0.13600904 -0.13600904 ... -0.13600904]] View as Array
> min = (float64) -0.26331944007432284
> max = (float64) 22.781571499789077
> shape = (tuple: 2) (1111, 520)
> dtype = (dtype: 0) float64
> size = (int) 577720
> array = (NdArrayItemsContainer) <pydevd_plugins.extensions.types.pydevd_plugin.numpy.types.NdArrayItemsContainer object at 0x0000023C555...

```

```

x_test_utm = (ndarray: (1111,)) [-7515.9167994 -7383.86722098 -7374.30207994 -7365.82488255 -7641.49930337 -7338.80721042 -7379.351...View as Array
 min = (float64) -7695.938754929929
 max = (float64) -7299.786516730872
 shape = (tuple: 1) 1111
 dtype = (dtype: 0) float64
 size = (int) 1111
 array = (NdArrayItemsContainer) <pydevd_plugins.extensions.types.pydevd_plugin_numpy_types.NdArrayItemsContainer object at 0x0000023C57B1ECD0>
x_train = (ndarray: (17928, 520)) [[-0.17125017 -0.17125017 -0.17125017 ... -0.17125017 -0.17125017, -0.17125017], [-0.16059846 -0.16059846...View as Array
x_val = (ndarray: (2009, 520)) [[-0.17119193 -0.17119193 -0.17119193 ... -0.17119193 -0.17119193, -0.17119193], [-0.17455084 -0.17455084 -C...View as Array
y_avg = (dict: 15) {'0-0': 4864957.72195798, '0-1': 4864957.706202947, '0-2': 4864958.3644638825, '0-3': 4864956.533051689, '0-4': nan, '1-0': 4864874 ... (显示)
y_test_utm = (ndarray: (1111,)) [4864889.66291669 4864839.74030718 4864846.5337223 4864842.82925374, 4864922.39917781 4864825.466 ...View as Array
 min = (float64) 4864748.01548531
 max = (float64) 4865017.364684202
 shape = (tuple: 1) 1111
 dtype = (dtype: 0) float64
 size = (int) 1111
 array = (NdArrayItemsContainer) <pydevd_plugins.extensions.types.pydevd_plugin_numpy_types.NdArrayItemsContainer object at 0x0000023C6736AF70>

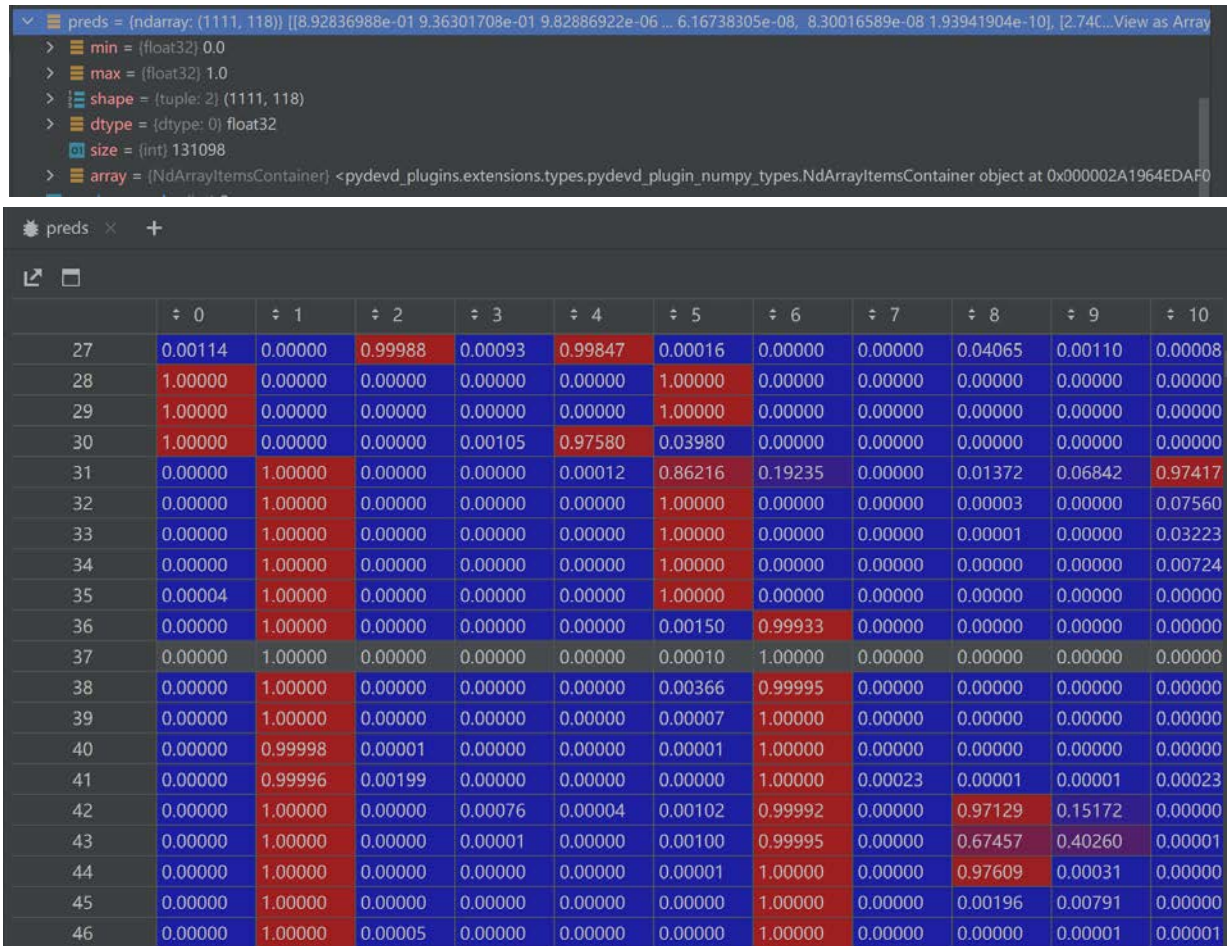
```

Model test:

Code:

```
1. preds = model.predict(test_AP_features, batch_size=batch_size)
```

Result:



The screenshot shows a Jupyter Notebook interface. The top part displays the output of a model prediction, showing a NumPy array of shape (1111, 118) with dtype float32. The bottom part shows a visualization of the array as a table with 11 columns (labeled 0 to 10) and 20 rows (labeled 27 to 46). The table contains numerical values, with some cells highlighted in red and others in blue.

|    | ÷ 0     | ÷ 1     | ÷ 2     | ÷ 3     | ÷ 4     | ÷ 5     | ÷ 6     | ÷ 7     | ÷ 8     | ÷ 9     | ÷ 10    |
|----|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|
| 27 | 0.00114 | 0.00000 | 0.99988 | 0.00093 | 0.99847 | 0.00016 | 0.00000 | 0.00000 | 0.04065 | 0.00110 | 0.00008 |
| 28 | 1.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 1.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 29 | 1.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 1.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 30 | 1.00000 | 0.00000 | 0.00000 | 0.00105 | 0.97580 | 0.03980 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 31 | 0.00000 | 1.00000 | 0.00000 | 0.00000 | 0.00012 | 0.86216 | 0.19235 | 0.00000 | 0.01372 | 0.06842 | 0.97417 |
| 32 | 0.00000 | 1.00000 | 0.00000 | 0.00000 | 0.00000 | 1.00000 | 0.00000 | 0.00000 | 0.00003 | 0.00000 | 0.07560 |
| 33 | 0.00000 | 1.00000 | 0.00000 | 0.00000 | 0.00000 | 1.00000 | 0.00000 | 0.00000 | 0.00001 | 0.00000 | 0.03223 |
| 34 | 0.00000 | 1.00000 | 0.00000 | 0.00000 | 0.00000 | 1.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00724 |
| 35 | 0.00004 | 1.00000 | 0.00000 | 0.00000 | 0.00000 | 1.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 36 | 0.00000 | 1.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00150 | 0.99933 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 37 | 0.00000 | 1.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00010 | 1.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 38 | 0.00000 | 1.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00366 | 0.99995 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 39 | 0.00000 | 1.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00007 | 1.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 40 | 0.00000 | 0.99998 | 0.00001 | 0.00000 | 0.00000 | 0.00001 | 1.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| 41 | 0.00000 | 0.99996 | 0.00199 | 0.00000 | 0.00000 | 0.00000 | 1.00000 | 0.00023 | 0.00001 | 0.00001 | 0.00023 |
| 42 | 0.00000 | 1.00000 | 0.00000 | 0.00076 | 0.00004 | 0.00102 | 0.99992 | 0.00000 | 0.97129 | 0.15172 | 0.00000 |
| 43 | 0.00000 | 1.00000 | 0.00000 | 0.00001 | 0.00000 | 0.00100 | 0.99995 | 0.00000 | 0.67457 | 0.40260 | 0.00001 |
| 44 | 0.00000 | 1.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00001 | 1.00000 | 0.00000 | 0.97609 | 0.00031 | 0.00000 |
| 45 | 0.00000 | 1.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 1.00000 | 0.00000 | 0.00196 | 0.00791 | 0.00000 |
| 46 | 0.00000 | 1.00000 | 0.00005 | 0.00000 | 0.00000 | 0.00000 | 1.00000 | 0.00000 | 0.00000 | 0.00001 | 0.00001 |

- Good localization performance in terms of the positioning error provided from the sample code.

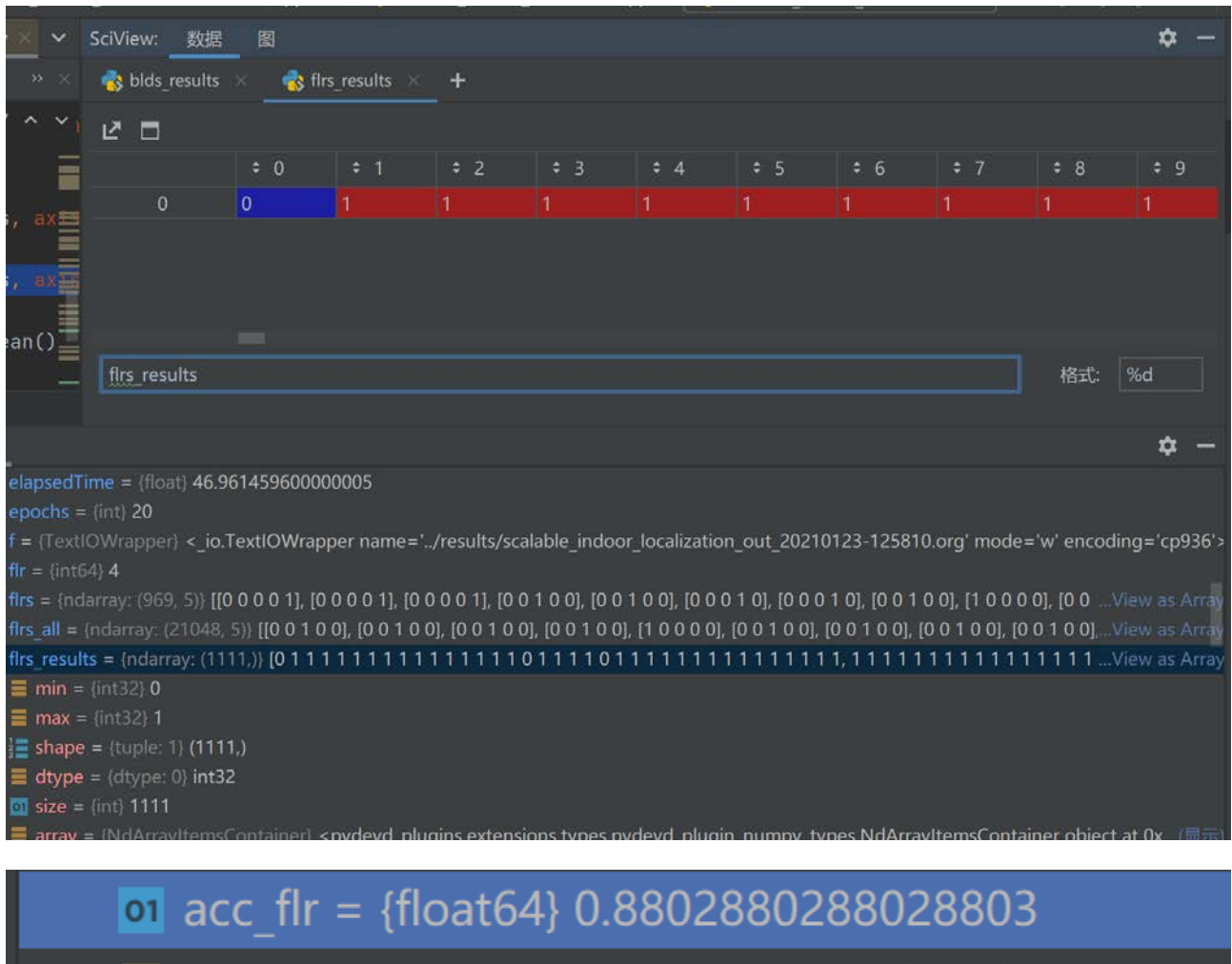
Good positioning is reflected in the code with various accuracy and errors. We can view the performance indicators in the last saved result.

\* Performance

- Accuracy (building): 9.873987e-01
- Accuracy (floor): 8.802880e-01
- Accuracy (building-floor): 8.721872e-01
- Location estimation failure rate (given the correct building/floor): 4.127967e-03
- Positioning error (meter): 1.171584e+01
- Positioning error (weighted; meter): 1.171584e+01



Result:



Accuracy (building-floor):

acc\_bf is a comprehensive evaluation index. If the index is higher, it means that the positioning performance of building and floor is very good.

Code:

```
1. acc_bf = (blds_results*firs_results).mean()
```

Result:

```
01 acc bf = {float64} 0.8721872187218722
```

Location estimation failure rate (given the correct building/floor):

Location estimation failure rate is the ratio of incorrect positioning to correct positioning. The

lower the ratio, the better the positioning of the algorithm.

Code:

```
1. if len(xs) > 0:
2. sum_pos_err += math.sqrt((np.mean(xs)-x_test_utm[i])**2 + (np.mean(ys)-
y_test_utm[i])**2)
3. sum_pos_err_weighted += math.sqrt((np.average(xs, weights=ws)-
x_test_utm[i])**2 + (np.average(ys,
4. weights=ws)-y_test_utm[i])**2)
5. else:
6. n_loc_failure += 1
7. key = str(np.argmax(blds[i])) + '-' + str(np.argmax(flrs[i]))
8. pos_err = math.sqrt((x_avg[key]-x_test_utm[i])**2 + (y_avg[key]-
y_test_utm[i])**2)
9. sum_pos_err += pos_err
10. sum_pos_err_weighted += pos_err
11. mean_pos_err = sum_pos_err / n_success
12. mean_pos_err_weighted = sum_pos_err_weighted / n_success
13. loc_failure = n_loc_failure / n_success
```

Result:

```
01 loc_failure = {float} 0.0041279669762641896
```

Positioning error:

Code:

```
1. if len(xs) > 0:
2. sum_pos_err += math.sqrt((np.mean(xs)-x_test_utm[i])**2 + (np.mean(ys)-
y_test_utm[i])**2)
3. sum_pos_err_weighted += math.sqrt((np.average(xs, weights=ws)-
x_test_utm[i])**2 + (np.average(ys,
4. weights=ws)-y_test_utm[i])**2)
5. else:
6. n_loc_failure += 1
7. key = str(np.argmax(blds[i])) + '-' + str(np.argmax(flrs[i]))
8. pos_err = math.sqrt((x_avg[key]-x_test_utm[i])**2 + (y_avg[key]-
y_test_utm[i])**2)
9. sum_pos_err += pos_err
```



```

10. sum_pos_err_weighted += pos_err
11. mean_pos_err = sum_pos_err / n_success

```

Positioning error (weighted; meter):

Code:

```

1. if len(xs) > 0:
2. sum_pos_err += math.sqrt((np.mean(xs)-x_test_utm[i])**2 + (np.mean(ys)-
y_test_utm[i])**2)
3. sum_pos_err_weighted += math.sqrt((np.average(xs, weights=ws)-
x_test_utm[i])**2 + (np.average(ys,
4. weights=ws)-y_test_utm[i])**2)
5. else:
6. n_loc_failure += 1
7. key = str(np.argmax(blds[i])) + '-' + str(np.argmax(flrs[i]))
8. pos_err = math.sqrt((x_avg[key]-x_test_utm[i])**2 + (y_avg[key]-
y_test_utm[i])**2)
9. sum_pos_err += pos_err
10. sum_pos_err_weighted += pos_err
11. mean_pos_err = sum_pos_err / n_success
12. mean_pos_err_weighted = sum_pos_err_weighted / n_success

```

## 4.3 Analysis and discussions.

- Performance comparison with the sample implementation.
- Results discussions.

From the code point of view, there are the following parameters that can be changed to achieve better positioning.

- Number of epochs
- Batch size
- SAE optimizer
- Classifier hidden layers
- Classifier dropout rate

## 1. Number of epochs

Epochs are defined as a single training iteration of all batches in the forward and backward propagation. This means that 1 cycle is a single forward and backward pass of the entire input data. Simply put, epochs refers to how many times the data will be "rounded" during the training process.[7]

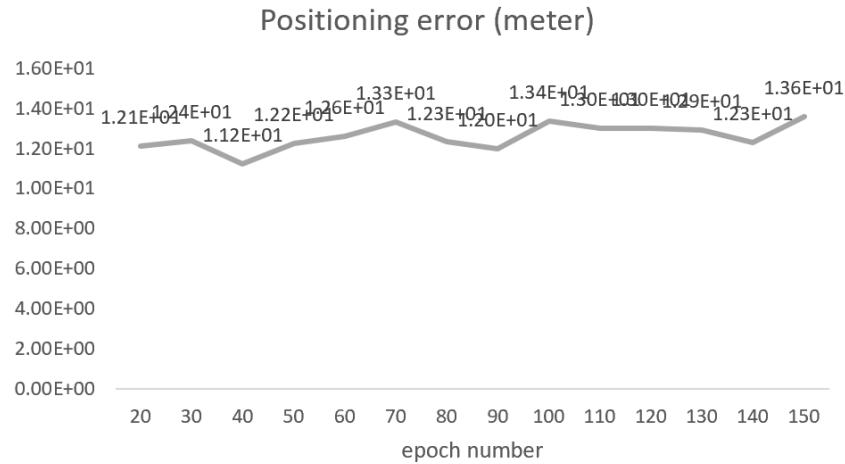
Code:

```
1. parser.add_argument(
2. "-E",
3. "--epochs",
4. help="number of epochs; default is 20",
5. default=100,
6. type=int)
```

When we fix other parameters, we will experiment with epoch number from 20 to 150.

Result:





After observation, we choose epoch number = 50 as the parameter

```
* Performance
- Accuracy (building): 9.837984e-01
- Accuracy (floor): 8.757876e-01
- Accuracy (building-floor): 8.658866e-01
- Location estimation failure rate (given the correct building/floor): 8.316008e-03
- Positioning error (meter): 1.224486e+01
- Positioning error (weighted; meter): 1.224486e+01
```

## 2. Batch size

Batch Size is the number of samples selected for one training. Before batch size is used, this means that when the network is training, it inputs all the data (the entire database) into the network at once, and then calculates their gradients for back propagation. Since the entire database is used when calculating the gradient, the calculated gradient direction is more accurate. But in this case, the difference between the calculated gradient values is huge, and it is difficult to use a global learning rate. Therefore, at this time, the gradient symbol-based training algorithm based on Rprop is generally used to update the gradient separately.<sup>7</sup>

In a database with a small sample size, it is feasible not to use Batch Size, and the effect is also very good. But once it is a large database, entering all the data into the network at one time will definitely cause a memory explosion. So put forward the concept of Batch Size.

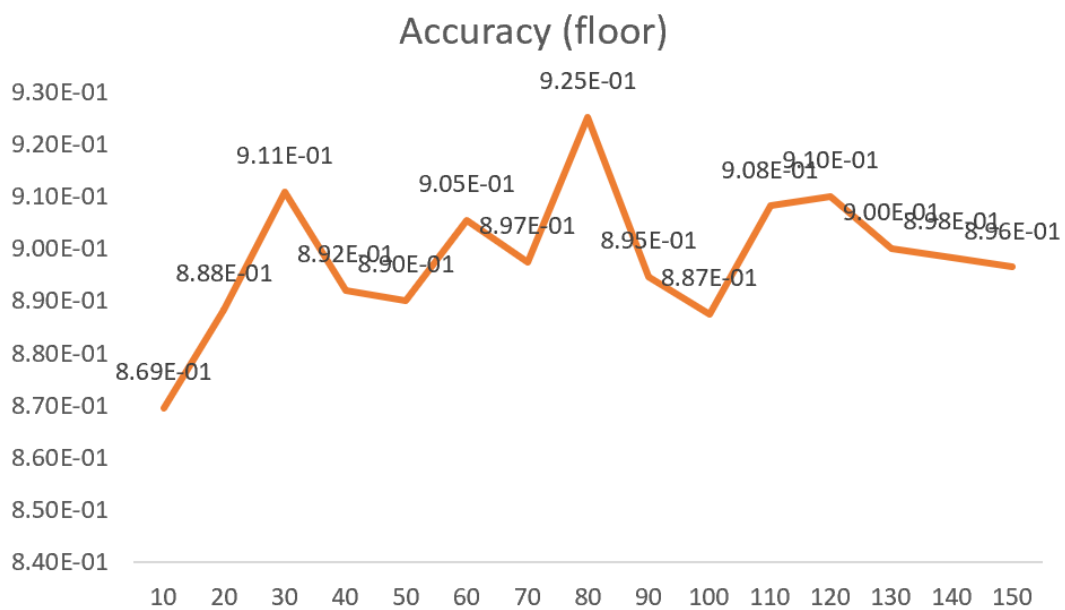
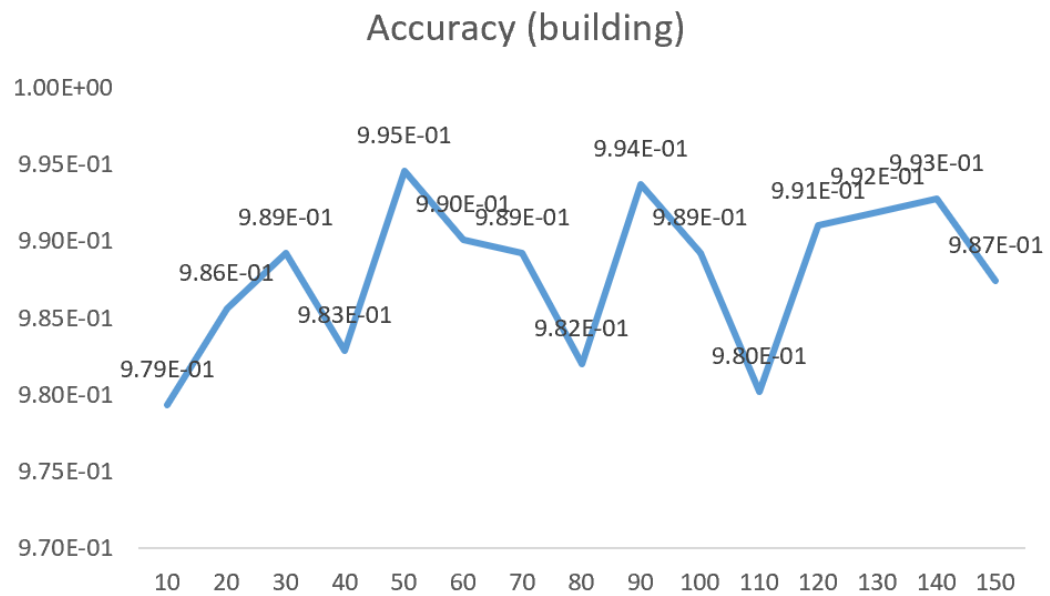
As the Batch Size increases, the gradient becomes accurate, and the Batch Size continues to increase. The gradient is already very accurate, and it is useless to increase the Batch Size.

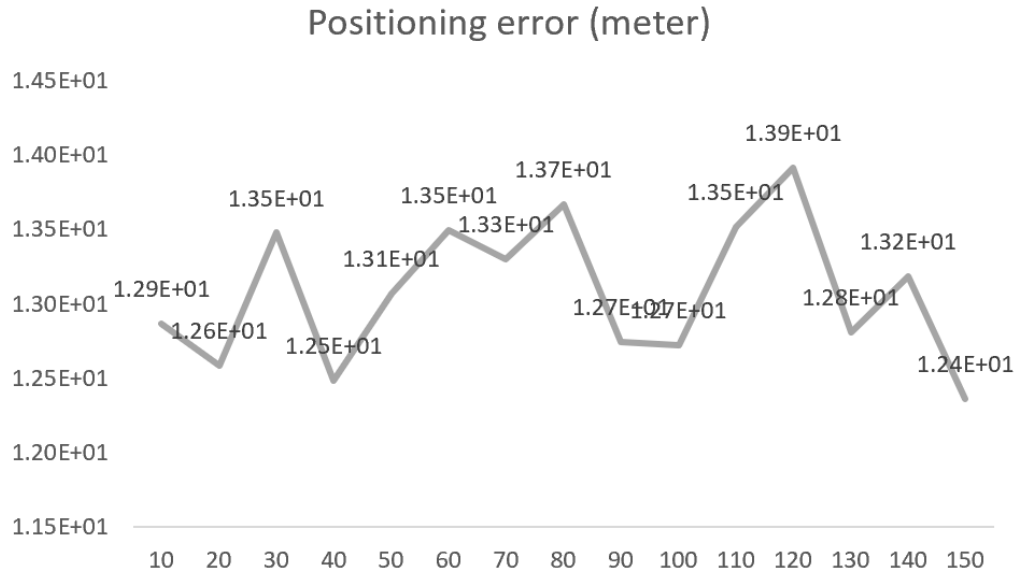


Code:

```
1. parser.add_argument(
2. "-B",
3. "--batch_size",
4. help="batch size; default is 10",
5. default=10,
6. type=int)
```

In order to get the best parameters, we start testing from 10 and continue testing to 150.





It can be observed that when batch size = 100, it is a better parameter, and the performance is as follows:

```
* Performance
- Accuracy (building): 9.981998e-01
- Accuracy (floor): 9.081908e-01
- Accuracy (building-floor): 9.072907e-01
- Location estimation failure rate (given the correct building/floor): 6.944444e-03
- Positioning error (meter): 1.241956e+01
- Positioning error (weighted; meter): 1.241956e+01
```

### 3. SAE optimizer

In machine learning, there are many optimization methods to try to find the optimal solution of the model.

The adaptive learning rate optimization algorithm is aimed at the learning rate of the machine learning model. The traditional optimization algorithm either sets the learning rate to a constant or adjusts the learning rate according to the number of training sessions. The possibility of other changes in the learning rate is greatly ignored. However, the learning rate has a significant impact on the performance of the model, so some strategies need to be adopted to find ways to update the learning rate to increase the training speed.

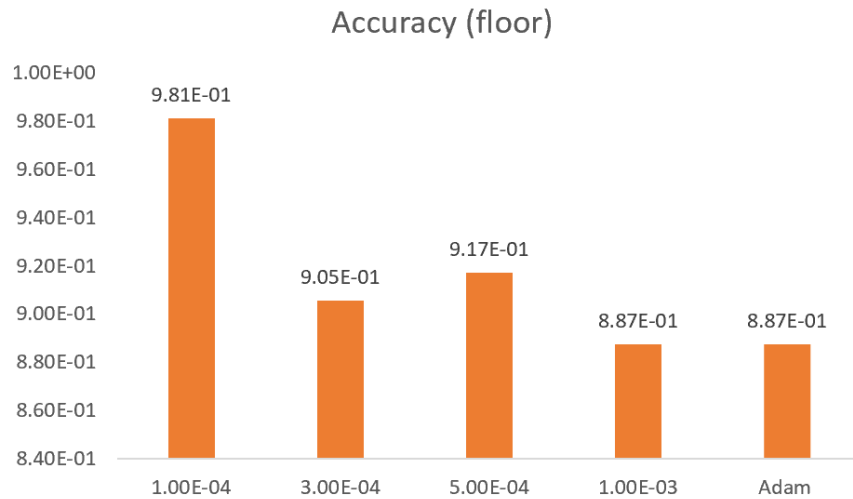
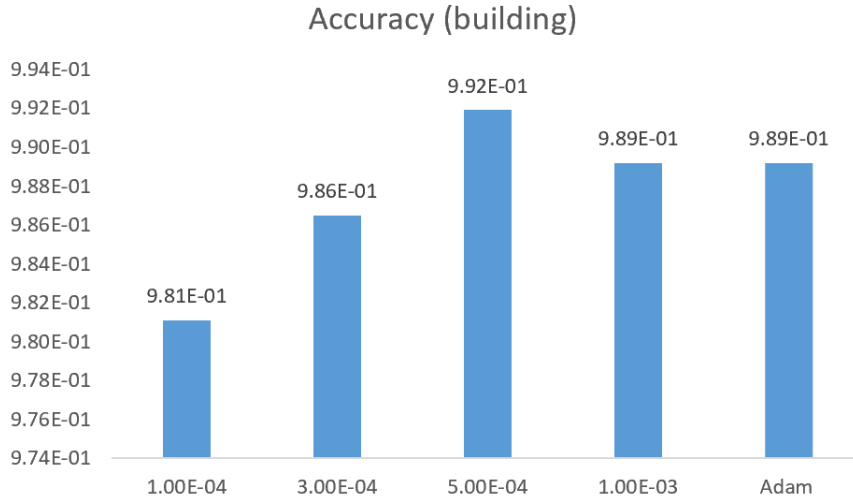
The current adaptive learning rate optimization algorithms mainly include: AdaGrad algorithm, RMSProp algorithm, Adam algorithm and AdaDelta algorithm.

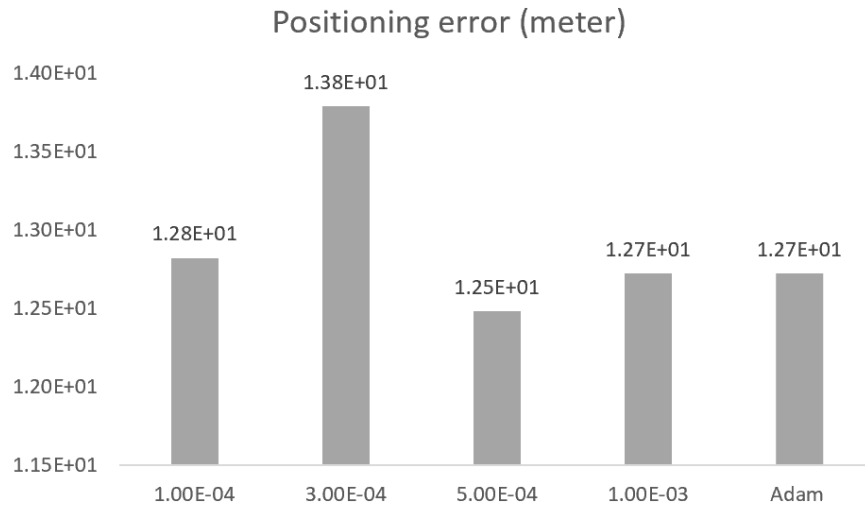
Adam update formula:

$$\begin{aligned}
 m_t &= \beta_1 m_{t-1} + (1 - \beta_1) g_t \\
 v_t &= \beta_2 v_{t-1} + (1 - \beta_2) g_t^2 \\
 \hat{m}_t &= \frac{m_t}{1 - \beta_1^t} \\
 \hat{v}_t &= \frac{v_t}{1 - \beta_2^t} \\
 \theta_{t+1} &= \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t
 \end{aligned}$$

In Adam's original paper and some deep learning frameworks, the default values are  $\eta=0.001$ ,  $\beta_1=0.9$ ,  $\beta_2=0.999$ , and  $\epsilon=1e-8$ . Among them,  $\beta_1$  and  $\beta_2$  are numbers close to 1, and  $\epsilon$  is to prevent division by 0.  $g_t$  represents the gradient.[8]

Here we compare the performance of the SAE\_OPTIMIZER default learning rate with  $1e-4$ ,  $3e-4$ ,  $5e-4$ , and  $1e-3$ .

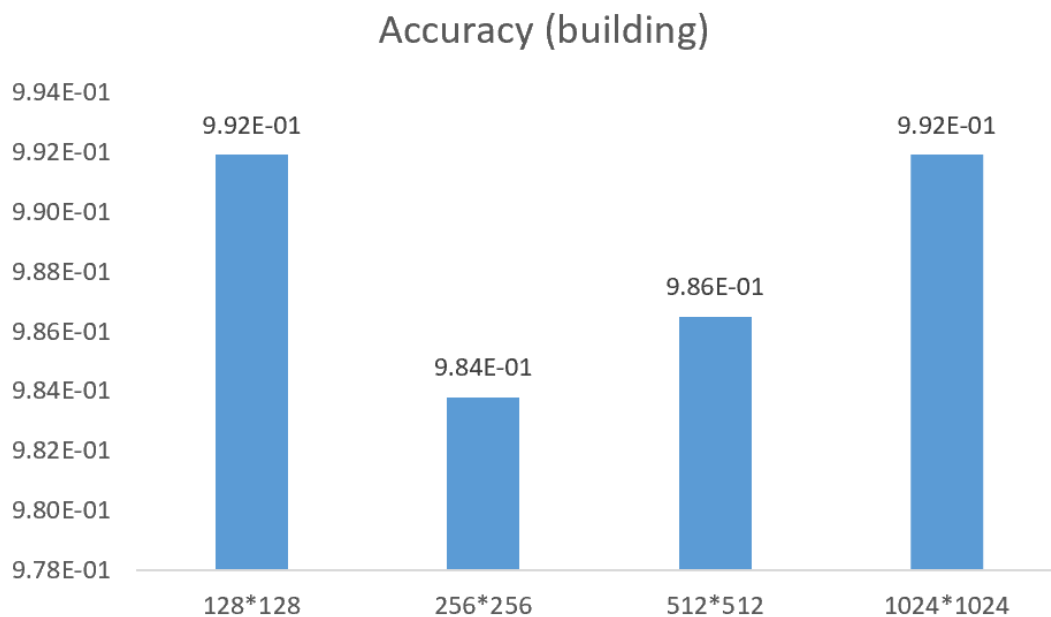


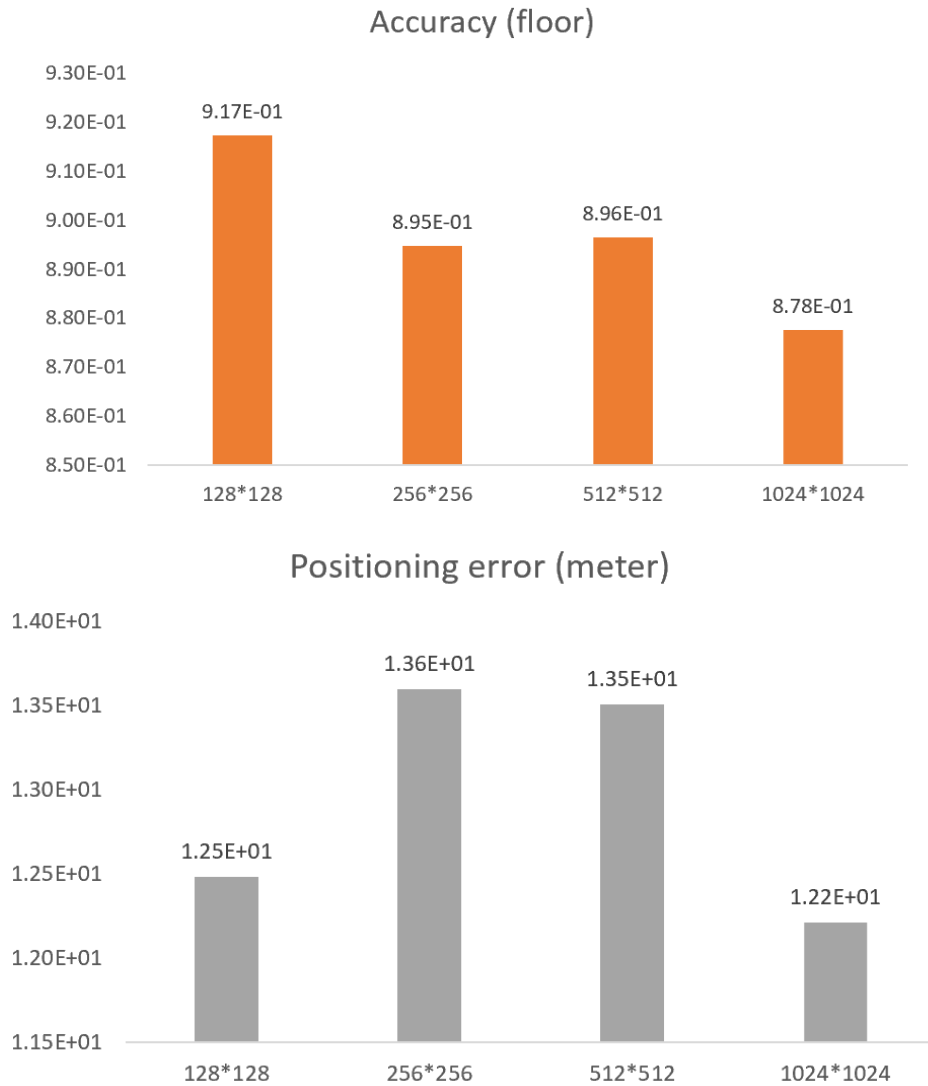


Observation shows that when the learning rate is  $5e-4$ , the performance is the best. At this time, the performance is:

```
* Performance
- Accuracy (building): 9.918992e-01
- Accuracy (floor): 9.171917e-01
- Accuracy (building-floor): 9.117912e-01
- Location estimation failure rate (given the correct building/floor): 1.382034e-02
- Positioning error (meter): 1.248317e+01
- Positioning error (weighted; meter): 1.248317e+01
```

#### 4. Classifier hidden layers





It can be observed that when the number of hidden layer nodes is 128\*128, the performance is the best. At this time, the performance is:

```
* Performance
- Accuracy (building): 9.918992e-01
- Accuracy (floor): 9.171917e-01
- Accuracy (building-floor): 9.117912e-01
- Location estimation failure rate (given the correct building/floor): 1.382034e-02
- Positioning error (meter): 1.248317e+01
- Positioning error (weighted; meter): 1.248317e+01
```

## 5. Classifier dropout rate

For any machine learning task, data is the key. If you are faced with a situation where the data is small and the model is complex, a fatal problem will be over-fitting. Over-fitting is specifically manifested in the loss function of the model on the training data. If the value is smaller, the prediction

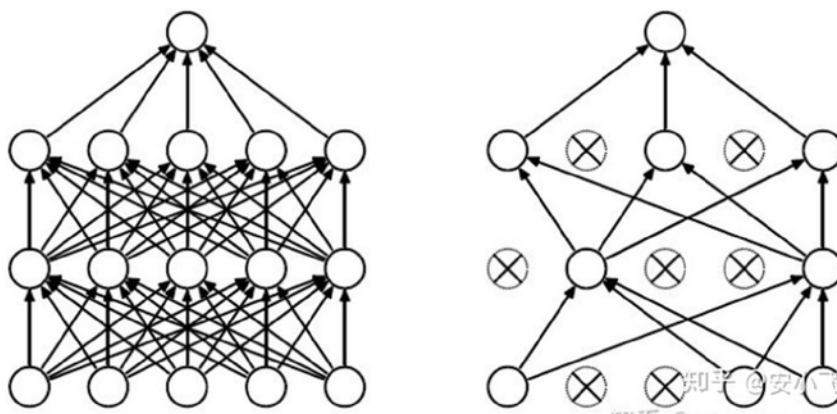
accuracy rate is higher; but the loss function on the test data is relatively large, and the prediction accuracy rate is lower. Therefore, the over-fitting model cannot be applied.

In order to solve the problem of overfitting, the method of model integration is generally adopted, that is, training multiple models to combine. At this time, the time-consuming training model becomes a big problem. Not only is it time-consuming to train multiple models, but it is also time-consuming to test multiple models. Especially for deep learning tasks, the problem is that the amount of calculation is huge, it is time-consuming, and it is easy to overfit.

In 2012, Hinton proposed Dropout in his paper "Improving neural networks by preventing co-adaptation of feature detectors".[9] When a complex feedforward neural network is trained on a small data set, it is easy to cause overfitting. In order to prevent overfitting, the performance of the neural network can be improved by preventing the joint action of feature detectors.

In 2012, Alex and Hinton used the Dropout algorithm in their paper "ImageNet Classification with Deep Convolutional Neural Networks" to prevent overfitting.[10] In addition, the AlexNet network model mentioned in this paper detonated the upsurge of neural network applications and won the 2012 image recognition competition, making CNN the core algorithm model for image classification.

The simple point of Dropout is: when we propagate forward, let the activation value of a certain neuron stop working with a certain probability  $p$ , which can make the model more generalized, because it will not rely too much on certain parts. The characteristics are as shown.



Suppose we want to train a neural network, the input is  $x$  and the output is  $y$ . The normal process is: we first propagate  $x$  forward through the network, and then propagate the error back to determine how to update the parameters for the network to learn. After using Dropout, the process becomes as

follows:

(1) First, randomly (temporarily) delete half of the hidden neurons in the network, and the input and output neurons remain unchanged (the dotted line in Figure 3 is part of the temporarily deleted neurons)

(2) Then propagate the input  $x$  forward through the modified network, and then propagate the resulting loss back through the modified network. After a small batch of training samples perform this process, the corresponding parameters ( $w$ ,  $b$ ) are updated according to the stochastic gradient descent method on the neurons that have not been deleted.

(3) Repeat this process.

Dropout has good performance in solving over-fitting problems,

(1) The effect of averaging: First return to the standard model, that is, there is no dropout. We use the same training data to train 5 different neural networks. Generally, we will get 5 different results. At this time, we can use "5 Take the average of the results or "the majority-winning voting strategy" to determine the final result. For example, the judgment result of 3 networks is the number 9, then it is very likely that the real result is the number 9, and the other two networks give wrong results. This "comprehensive average" strategy can usually effectively prevent overfitting. Because different networks may produce different overfitting, averaging may make some "opposite" fits cancel each other out. Dropout different hidden neurons is similar to training different networks. Randomly deleting half of the hidden neurons results in a different network structure. The entire dropout process is equivalent to taking the average of many different neural networks. Different networks produce different over-fitting, and some "reverse" fittings cancel each other out to reduce over-fitting as a whole.

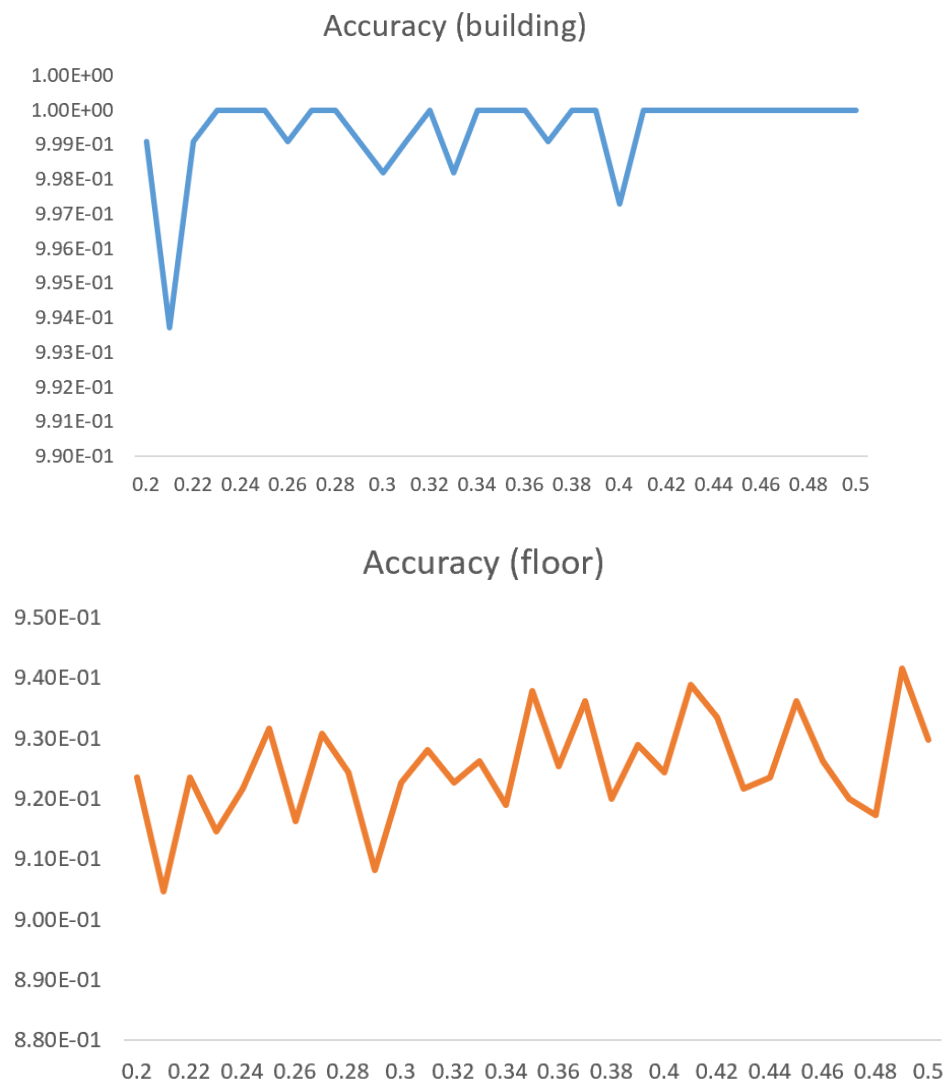
(2) Reduce the complex co-adaptation relationship between neurons: Because the dropout program causes two neurons to not necessarily appear in a dropout network every time. In this way, the update of weights no longer depends on the joint action of implicit nodes with fixed relationships, which prevents certain features from being effective only under other specific features. Force the network to learn more robust features, which also exist in random subsets of other neurons. In other words, if our neural network is making a certain prediction, it should not be too sensitive to some specific clue fragments. Even if the specific clue is lost, it should be able to learn some common

features from many other clues. From this perspective, dropout is a bit like L1, L2 regularity, reducing the weight makes the network more robust to the loss of specific neuron connections.

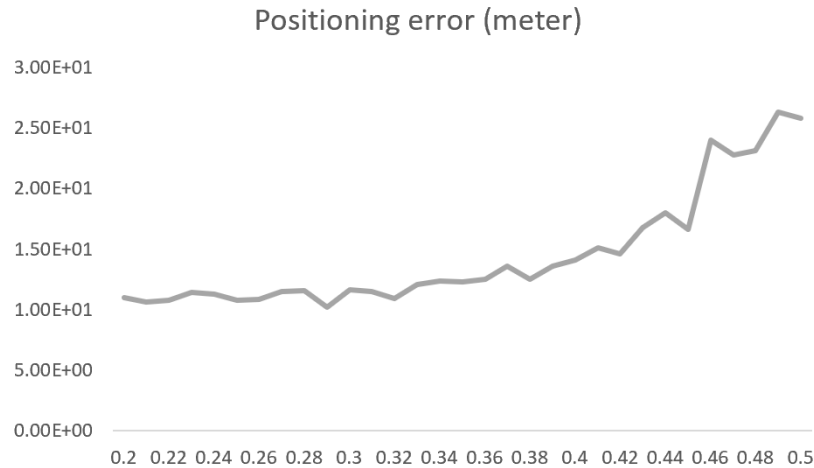
(3) Dropout is similar to the role of gender in biological evolution: species tend to adapt to this environment in order to survive, and environmental mutations will make it difficult for species to respond in time. The emergence of gender can breed variants that adapt to new environments. Effectively prevent overfitting, that is, avoid the extinction that species may face when the environment changes.

For a neuron, there are two states of retention and loss. It is natural to set the same probability for both states, that is, dropout(0.5)[7], but in fact, in many cases, the probability of drop in the early stage of a deep network Will be relatively small, around 0.1~0.3.

Therefore, we adjusted the parameters from 0.2 to 0.5 on the basis of the original parameters.







Through observation, we find the following dropout: 0.25——0.29

After analyzing all the above parameters, I got the following results after analysis:

```
#+STARTUP: showall
```

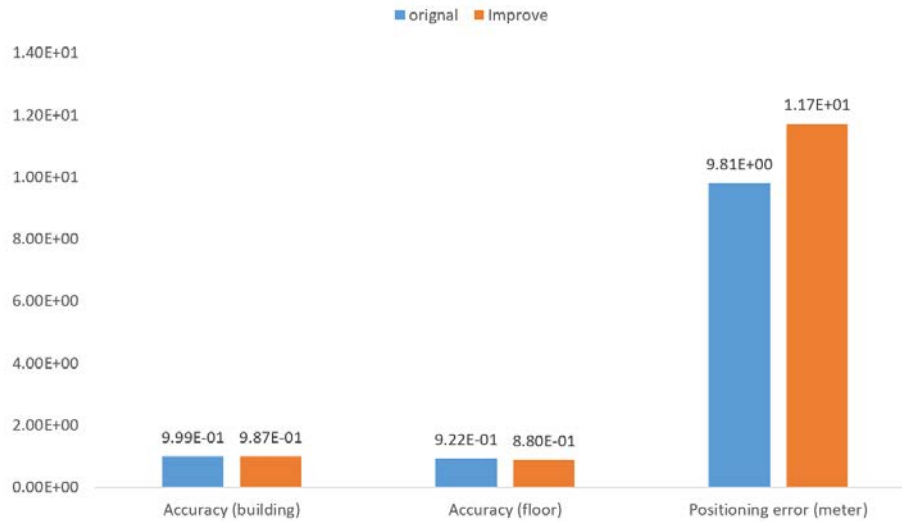
```
* System parameters
```

- Numpy random number seed: 0
- Ratio of training data to overall data: 0.90
- Number of epochs: 50
- Batch size: 100
- Number of neighbours: 1
- Scaling factor for threshold: 0.00
- SAE hidden layers: 256-128-64-128-256
- SAE activation: relu
- SAE bias: False
- SAE optimizer: adam
- SAE loss: mse
- Classifier hidden layers: 1024-1024
- Classifier hidden layer activation: relu
- Classifier bias: False
- Classifier optimizer: adam
- Classifier loss: binary\_crossentropy
- Classifier dropout rate: 0.27

```
* Performance
```

- Accuracy (building): 9.990999e-01
- Accuracy (floor): 9.216922e-01
- Accuracy (building-floor): 9.207921e-01
- Location estimation failure rate (given the correct building/floor): 2.150538e-02
- Positioning error (meter): 9.809429e+00
- Positioning error (weighted; meter): 9.809429e+00

After debugging, it was found that all the performances are not very well improved. Compared with the initial data, the effect is as follows:



Others There are some parameter settings with better performance, but not all performance indicators are good, so here is only the display of the results, not too much analysis.

```
#+STARTUP: showall
* System parameters
- Numpy random number seed: 0
- Ratio of training data to overall data: 0.90
- Number of epochs: 50
- Batch size: 100
- Number of neighbours: 1
- Scaling factor for threshold: 0.00
- SAE hidden layers: 256-128-64-128-256
- SAE activation: relu
- SAE bias: False
- SAE optimizer: <tensorflow.python.keras.optimizer_v2.adam.Adam object at 0x000001A510E17760>
- SAE loss: mse
- Classifier hidden layers: 512-512
- Classifier hidden layer activation: relu
- Classifier bias: False
- Classifier optimizer: adam
- Classifier loss: binary_crossentropy
- Classifier dropout rate: 0.27
* Performance
- Accuracy (building): 9.990999e-01
- Accuracy (floor): 8.982898e-01
- Accuracy (building-floor): 8.973897e-01
- Location estimation failure rate (given the correct building/floor): 4.012036e-03
- Positioning error (meter): 9.642798e+00
- Positioning error (weighted; meter): 9.642798e+00
```

```

#+STARTUP: showall
* System parameters
- Numpy random number seed: 0
- Ratio of training data to overall data: 0.90
- Number of epochs: 50
- Batch size: 100
- Number of neighbours: 1
- Scaling factor for threshold: 0.00
- SAE hidden layers: 256-128-64-128-256
- SAE activation: relu
- SAE bias: False
- SAE optimizer: adam
- SAE loss: mse
- Classifier hidden layers: 256-256
- Classifier hidden layer activation: relu
- Classifier bias: False
- Classifier optimizer: adam
- Classifier loss: binary_crossentropy
- Classifier dropout rate: 0.27
* Performance
- Accuracy (building): 1.000000e+00
- Accuracy (floor): 9.315932e-01
- Accuracy (building-floor): 9.315932e-01
- Location estimation failure rate (given the correct building/floor): 5.797101e-03
- Positioning error (meter): 1.035785e+01
- Positioning error (weighted; meter): 1.035785e+01

```

## 5 Conclusion

This experiment is based on the UJIIndoorLoc data set, using DNN technology, the TensorFlow framework and the Keras open source library to deploy and implement an indoor positioning algorithm based on RSS signal strength. On the basis of the Lab3 demo code, by adjusting the parameters, we can obtain a significant improvement, building and floor hit rates achieved for the UJIIndoorLoc dataset are 100% and 93.1%. But there is still a certain gap with the current best performance[11], in terms of positioning errors, there is still much room for improvement in terms of performance indicators.

We can consider using the RNN network[12] as a positioning network result in subsequent experiments, and you can try to use Batch Normalization technology.

## REFERENCES

- [1] J. Torres-Sospedra et al., "UJIIndoorLoc: A new multi-building and multi-floor database for WLAN fingerprint-based indoor localization problems," 2014 International Conference on Indoor Positioning and Indoor Navigation (IPIN), Busan, 2014, pp. 261-270, doi: 10.1109/IPIN.2014.7275492.
- [2] Kim, K., Lee, S. & Huang, K. A scalable deep neural network architecture for multi-building and multi-floor indoor localization based on Wi-Fi fingerprinting. *Big Data Anal* 3, 4 (2018). <https://doi.org/10.1186/s41044-018-0031-2>
- [3] Vincent, P., Larochelle, H., Lajoie, I., Bengio, Y., Manzagol, P. A., & Bottou, L. (2010). Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of machine learning research*, 11(12).
- [4] <https://machinelearningmastery.com/how-to-one-hot-encode-sequence-data-in-python/>
- [5] Epelbaum, T. (2017). Deep learning: Technical introduction. arXiv preprint arXiv:1709.01412.
- [6] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov (2014) : Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, pages 1929 – 1958, 2014.
- [7] Brownlee, J. (2018). What is the Difference Between a Batch and an Epoch in a Neural Network?. *Deep Learning; Machine Learning Mastery*: Vermont, VIC, Australia.
- [8] Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980.
- [9] Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. R. (2012). Improving neural networks by preventing co-adaptation of feature detectors. arXiv preprint arXiv:1207.0580.
- [10] Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 1097-1105.
- [11] Moreira, A., Nicolau, M. J., Meneses, F., & Costa, A. (2015, October). Wi-Fi fingerprinting in the real world-RTLS@ UM at the EvAAL competition. In *2015 International Conference on Indoor Positioning and Indoor Navigation (IPIN)* (pp. 1-10). IEEE.
- [12] Kim, K. S., Wang, R., Zhong, Z., Tan, Z., Song, H., Cha, J., & Lee, S. (2018). Large-scale location-aware services in access: Hierarchical building/floor classification and location estimation using Wi-Fi fingerprinting based on deep neural networks. *Fiber and Integrated Optics*, 37(5), 277-289.