House Sale Price Prediction

電子三甲 105360046 蕭賢傑

import

```
1 import datetime
2 import time as time
3 import pandas as pd
4 import numpy as np
5 import matplotlib.pyplot as plt
6 import tensorflow as tf
7 import keras
8 from keras.models import Sequential
9 from keras.layers import Dense
10 from keras import initializers
11 from keras import optimizers
12
```

Set parameter

```
13 output = False
14 use_gpu = False
15 train_valid = False
16
17 verbose = 2
18 batch_size = 32
19 epochs = 1000000
20 prediction_target_name = 'price'
21 hidden_unit = 64
22 loss = 'mean_absolute_error'
23
```

Setup device

```
24 gpu_options = tf.GPUOptions(allow_growth=True)
25 sess = tf.Session(config=tf.ConfigProto(
26         gpu_options=gpu_options,
27         device_count={'GPU':1 if use_gpu else 0, 'CPU':4}))
28 keras.backend.set_session(sess)
```

Read data

```
30 trainFile = pd.read_csv('./train-v3.csv', index_col=0)
31 validFile = pd.read_csv('./valid-v3.csv', index_col=0)
32 testFile = pd.read_csv('./test-v3.csv', index_col=0)
33
34 trainFile.index = np.linspace(0, trainFile.shape[0]-1, trainFile.shape[0], dtype=int)
35 validFile.index = np.linspace(0, validFile.shape[0]-1, validFile.shape[0], dtype=int)
36 testFile.index = np.linspace(0, testFile.shape[0]-1, testFile.shape[0], dtype=int)
```

Remove specify data

```
38 for i in range(trainFile.shape[0]):
39     if (trainFile.at[i, 'price'] > 3000000):
40         trainFile = trainFile.drop(i, axis=0)
41         i -= 1;
42 
43 if train_valid:
44     for i in range(validFile.shape[0]):
45         if (validFile.at[i, 'price'] > 3000000):
46             validFile = validFile.drop(i, axis=0)
47             i -= 1;
48 
49 trainFile = trainFile.dropna()
50 validFile = validFile.dropna()
51 testFile = testFile.dropna()
```

Split data to X and Y

```
53 X_train = trainFile.drop(columns=prediction_target_name)
54 Y_train = trainFile[prediction_target_name]
55
56 X_valid = validFile.drop(columns=prediction_target_name)
57 Y_valid = validFile[prediction_target_name]
58
59 X_test = testFile
```

Sandardizing data

```
61 X_train_stats = X_train.describe()
62 X_train_stats = X_train_stats.transpose()
63
64 Y_train_mean = Y_train.mean()
65 Y_train_std = Y_train.std()
66
67 def normX(x):
68   return (x - X_train_stats['mean']) / X_train_stats['std']
69
70 def normY(y):
71   return (y - Y_train_mean) / Y_train_std
72
73 def inormY(y):
74   return y * Y_train_std + Y_train_mean
75
76 X_train = normX(X_train)
77 X_valid = normX(X_valid)
78 X_test = normX(X_test)
79
80 Y_train = normY(Y_train)
81 Y_valid = normY(Y_valid)
```

Optimizer initializer and activation

```
95 optimizer = optimizers.adam(lr=0.001, beta_1=0.9, beta_2=0.999, epsilon=1e-24, decay=0, amsgrad=False)
96
97 kernel_initializer = initializers.uniform(minval=-0.05, maxval=0.05, seed=None)
98 activation = 'relu'
```

Callbacks

```
class RestoreBestWeightsFinal(keras.callbacks.Callback):
    def __init__(self,
                 min_delta=0,
                 mode='auto',
                 baseline=None):
        super(RestoreBestWeightsFinal, self).__init__()
        self.min_delta = min_delta
        self.best_weights = None

        if mode not in ['auto', 'min', 'max']:
            mode = 'auto'

        if mode == 'min':
            self.monitor_op = np.less
        elif mode == 'max':
            self.monitor_op = np.greater
        else:
            self.monitor_op = np.less

        if self.monitor_op == np.greater:
            self.min_delta *= 1
        else:
            self.min_delta *= -1

    def on_train_begin(self, logs=None):
        # Allow instances to be re-used
        self.best = np.Inf if self.monitor_op == np.less else -np.Inf

    def on_train_end(self, logs=None):
        if self.best_weights is not None:
            self.model.set_weights(self.best_weights)

    def on_epoch_end(self, epoch, logs=None):
        val_current = logs.get('val_loss')
        if val_current is None:
            return

        if self.monitor_op(val_current - self.min_delta, self.best):
            self.best = val_current
            self.best_weights = self.model.get_weights()

callbacks = []
callbacks.append(keras.callbacks.EarlyStopping(monitor='val_loss', patience=10))
callbacks.append(RestoreBestWeightsFinal())
```

Model

```
145 model = Sequential()
146 model.add(Dense(int(hidden_unit), activation=activation, kernel_initializer=kernel_initializer, input_dim=X_train.shape[1]))
147 model.add(Dense(int(hidden_unit/2), activation=activation, kernel_initializer=kernel_initializer))
148 model.add(Dense( 1, activation=None))
149 model.compile(optimizer=optimizer, loss=loss)
150 model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_22 (Dense)             (None, 64)                1408
_________________________________________________________________
dense_23 (Dense)             (None, 32)                2080
_________________________________________________________________
dense_24 (Dense)             (None, 1)                 33
=================================================================
Total params: 3,521
Trainable params: 3,521
Non-trainable params: 0
```

Fit model and predict

```
151
152 t = time.time()
153 history_o = model.fit(X_train, Y_train, validation_data= None if train_valid else (X_valid, Y_valid),
154                       batch_size=batch_size, epochs=epochs, verbose=verbose, callbacks=callbacks)
155 elapsed = time.time() - t
156
157 history = pd.DataFrame(history_o.history)
158
159 Y_test = model.predict(X_test)
160
161 Y_test = inormY(Y_test)
162 history = history * Y_train_std
163
164 history['epoch'] = history_o.epoch
```

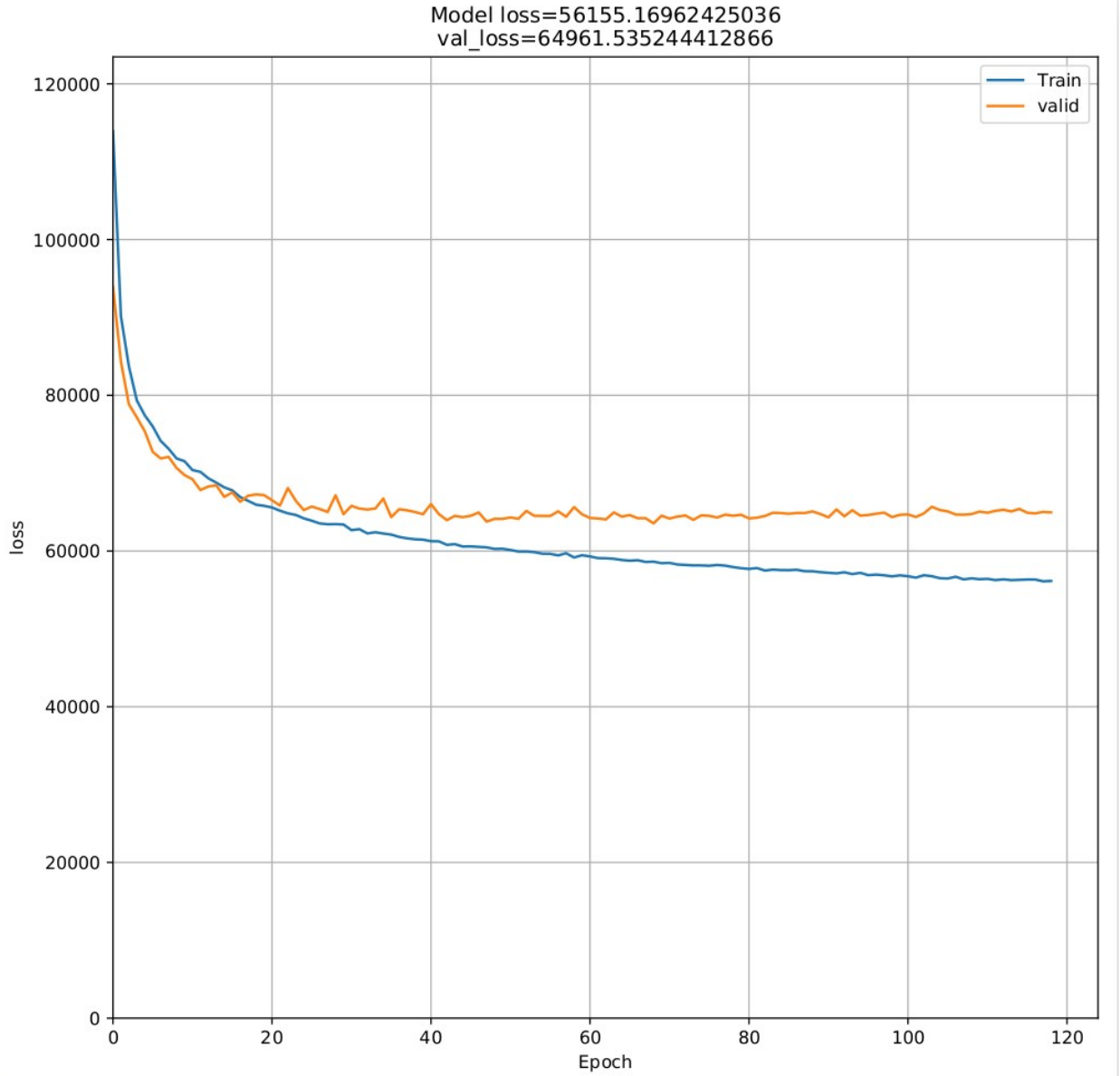
Plot result

```
176 f = plt.figure(figsize=(10,10));
177 train_valid = False
178 plt.plot(history['loss'])
179 if not train_valid:
180     plt.plot(history['val_loss'])
181 plt.title('elapsed='+str(elapsed)+'s\n'
182           +'loss='+str(history['loss'].values[-1])+
183           (train_valid and ' ' or '\nval_loss='+str(history['val_loss'].values[-1])))
184 plt.ylabel('loss')
185 plt.xlabel('Epoch')
186 plt.ylim(0, history['loss'].mean()*2)
187 plt.xlim(left=0)
188 plt.grid()
189 plt.legend(train_valid and 'Train' or ['Train', 'valid'], loc='upper right')
190 plt.show()
191
```

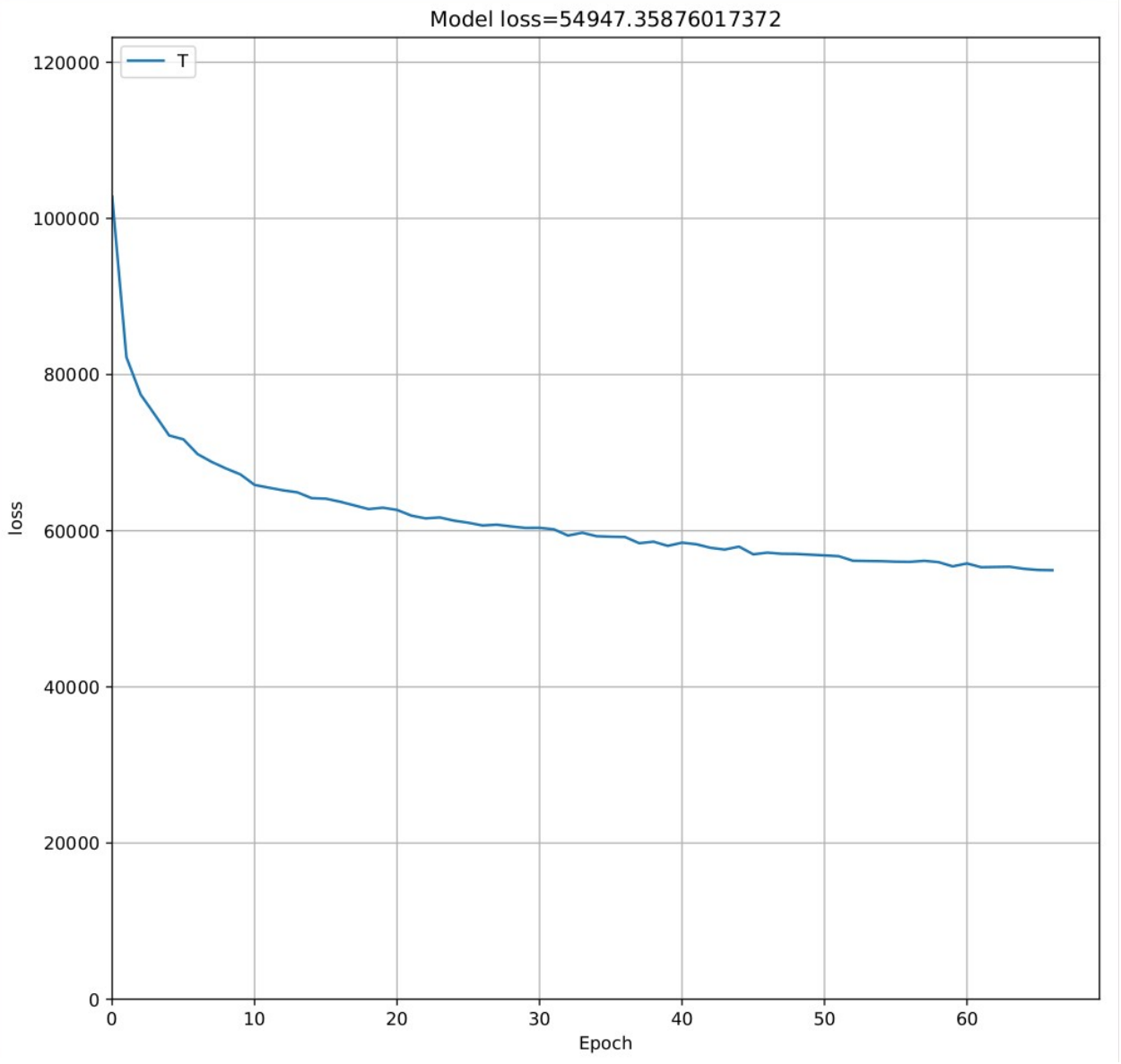
Save result, figure and model

```
192 if output:
193     Y_test_csv_format = pd.DataFrame(Y_test,
194              index=np.linspace(1, Y_test.shape[0], Y_test.shape[0], dtype=int),
195              columns=[prediction_target_name])
196     Y_test_csv_format.to_csv(filename+'.csv', index_label='id')
197     f.savefig(filename+'.pdf', bbox_inches='tight')
198     model_json = model.to_json()
199     with open(filename+'.json', 'w') as json_file:
200         json_file.write(model_json)
201     model.save_weights(filename+'.h5')
202
```

History


Model loss=56155.16962425036
val_loss=64961.535244412866
Train
valid
loss
Epoch
0
20000
40000
60000
80000
100000
120000
0
20
40
60
80
100
120

Put valid-data into train-data



改進方式

1. 將跟價錢最無關的幾個欄位刪除
2. 將欄位與價錢線性化
3. 加入 Dropout, BatchNormalization 等
4. 嘗試使用不同的 Optimizer