Keras

Example 4 – Keras for regression

Example 4 – Boston dataset

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
```

```
import tensorflow as tf
import random as python_random
```

```
from keras.models import Sequential
from keras.layers import Dense
```

Example 4 – Boston dataset

```
df0 = pd.read_csv('Boston.csv')
df0.shape
```

```
(506, 14)
```

```
df0[:5]
```

	crim	zn	indus	chas	nox	rm	age	dis	rad	tax	ptratio	black	lstat	medv
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222	18.7	396.90	5.33	36.2

Example 4 – Boston dataset

```
df = df0.values
X = df[:,0:13]
y = df[:,13]
```

```
# Reserve test set for performance evaluation
```

```
X_train,X_test,y_train,y_test = train_test_split(X,y,random_state=7,
                                                 test_size = 0.20)
```

```
scaler = StandardScaler()
scaler.fit(X_train)

Xtrain_scaled = scaler.transform(X_train)
Xtest_scaled = scaler.transform(X_test)
```

```
X_train.shape
```

```
(404, 13)
```

Example 4 – Boston dataset

```
# to have reproducible results
```

```
j = 999
# for starting Numpy generated random numbers
np.random.seed(j)
# for starting core Python generated random numbers
python_random.seed(j)
# for starting tensorflow random number generation
tf.random.set_seed(j)
```

```
# An input layer with 13 nodes (one for each predictor)
# two hidden layers with 64 nodes
# try 10 epochs, then will increase to 50
# for regression problems, use output layer with one node, always
```

```
model = Sequential()
model.add(Dense(64,activation = 'relu', input_shape=(13,)))
model.add(Dense(64,activation = 'relu'))
model.add(Dense(1))
model.compile(optimizer = 'rmsprop', loss = 'mse', metrics=['mae'])
model.fit(Xtrain_scaled,y_train,epochs =10, batch_size = 1);
```

Example 4 – Boston dataset

```
model = Sequential()
model.add(Dense(64,activation = 'relu', input_shape=(13,)))
model.add(Dense(64,activation = 'relu'))
model.add(Dense(1))
model.compile(optimizer = 'rmsprop', loss = 'mse', metrics=['mae'])
model.fit(Xtrain_scaled,y_train,epochs =10, batch_size = 1)
```

```
Epoch 1/10
404/404 [==============================] - 1s 1ms/step - loss: 165.145
5 - mae: 9.1668
Epoch 2/10
404/404 [==============================] - 0s 1ms/step - loss: 21.1889
- mae: 3.1907
Epoch 3/10
404/404 [==============================] - 1s 2ms/step - loss: 15.6568
- mae: 2.7467

...
```

```
mse, mae = model.evaluate(Xtest_scaled,y_test,verbose=0)
print(mse,',',mae)
```

```
27.150823985829074 , 3.040967702865600
```

Example 4 – Boston dataset

1 input layer

2 hidden layers

```
model = Sequential()
model.add(Dense(64,activation = 'relu', input_shape=(13,)))
model.add(Dense(64,activation = 'relu'))
model.add(Dense(1))
model.compile(optimizer = 'rmsprop', loss = 'mse', metrics=['mae'])
model.fit(Xtrain_scaled,y_train,epochs =10, batch_size = 1)
```

1 output layer

```
Epoch 1/10
404/404 [==============================] - 1s 1ms/step - loss: 165.145
5 - mae: 9.1668
Epoch 2/10
404/404 [==============================] - 0s 1ms/step - loss: 21.1889
- mae: 3.1907
Epoch 3/10
404/404 [==============================] - 1s 2ms/step - loss: 15.6568
- mae: 2.7467

...
```

```
mse, mae = model.evaluate(Xtest_scaled,y_test,verbose=0)
print(mse,',',mae)
```

```
27.150823985829074 , 3.040967702865600
```

Example 4 – Boston dataset

```
model = Sequential()
model.add(Dense(64,activation = 'relu', input_shape=(13,)))
model.add(Dense(64,activation = 'relu'))
model.add(Dense(1))
model.compile(optimizer = 'rmsprop', loss = 'mse', metrics=['mae'])
model.fit(Xtrain_scaled,y_train,epochs =50, batch_size = 1,verbose = 0)
```

```
mse, mae = model.evaluate(Xtest_scaled,y_test,verbose=0)
print(mse,',',mae)
```

```
22.646251933247434 , 2.7271087169647217
```

Example 4 – K-fold Cross Validation

```
def build_model():
    model = Sequential()
    model.add(Dense(64,activation = 'relu', input_shape=(13,)))
    model.add(Dense(64,activation = 'relu'))
    model.add(Dense(1))
    model.compile(optimizer = 'rmsprop', loss = 'mse', metrics=['mae'])
    return model
```

```
# use 4 folds
k = 4
```

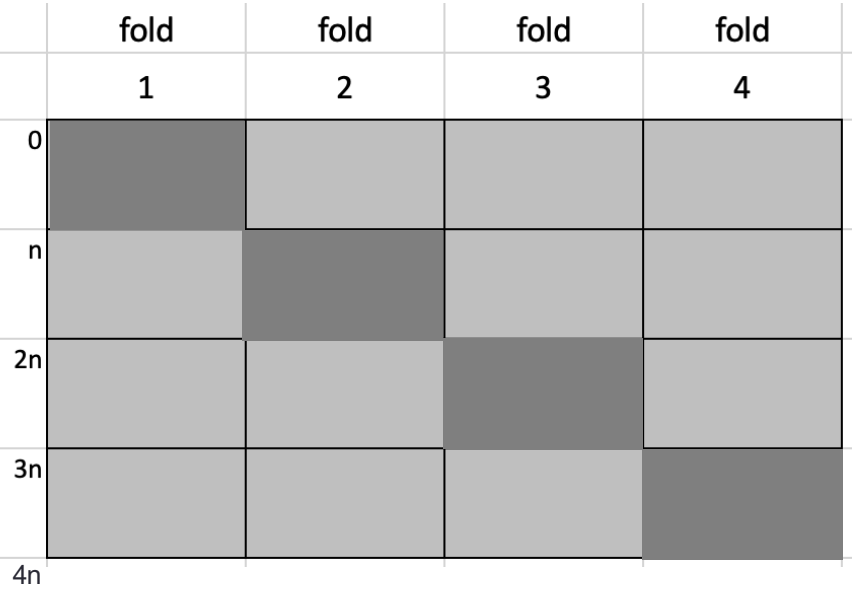
```
# number of observations per fold
n = len(Xtrain_scaled)//k
n
```

round down

```
101
```

Example 4 – K-fold Cross Validation

fold i	Validation set	Train set
0	[0, n)	[n, 4n]
1	[n, 2n)	[0, n), [2n, 4n]
2	[2n,3n)	[0,2n), [3n, 4n]
3	[3n,4n)	[0, 3n)



Example 4 – K-fold Cross Validation

n = 101

```
for i in range(k):
    print ('processing fold #', i)

    # Select ith fold test set
    X_val = Xtrain_scaled[i*n:(i+1)*n]
    y_val = y_train[i*n:(i+1)*n]
```

fold *i*	Validation set
0	[0, n)
1	[n, 2n)
2	[2n,3n)
3	[3n,4n)

Example 4 – K-fold Cross Validation

n = 101

```
for i in range(k):
```

fold *i*	Train set
0	[n, 4n]
1	[0, n), [2n, 4n]
2	[0,2n), [3n, 4n]
3	[0, 3n)

```
    # Create ith fold train set
    X_fold = np.concatenate([Xtrain_scaled[:i*n],
                             Xtrain_scaled[(i+1)*n:]], axis = 0)
    y_fold = np.concatenate([y_train[:i*n],
                             y_train[(i+1)*n:]], axis = 0)
```

Example 4 – K-fold Cross Validation

n = 101

```
for i in range(k):
```

```
    # Train ith fold
    model.fit(X_fold,y_fold, epochs = 50, batch_size = 1,verbose = 0)
    # Test ith fold
    mse, mae = model.evaluate(X_val,y_val,verbose = 0)
    scores.append(mae)
```

Example 4 – K-fold Cross Validation

n = 101

```
for i in range(k):
    print ('processing fold #', i)

    # Select ith fold test set
    X_val = Xtrain_scaled[i*n:(i+1)*n]
    y_val = y_train[i*n:(i+1)*n]

    # Create ith fold train set
    X_fold = np.concatenate([Xtrain_scaled[:i*n],
                             Xtrain_scaled[(i+1)*n:]], axis = 0)
    y_fold = np.concatenate([y_train[:i*n],
                             y_train[(i+1)*n:]], axis = 0)
    model = build_model()

    # Train ith fold
    model.fit(X_fold,y_fold, epochs = 50, batch_size = 1,verbose = 0)
    # Test ith fold
    mse, mae = model.evaluate(X_val,y_val,verbose = 0)
    scores.append(mae)
```

Example 4 – K-fold Cross Validation

```
scores
```

```
[2.194908380508423, 2.8107619285583496, 2.281386613845825, 2.33
918335]
```

```
np.mean(scores)
```

```
2.404637038707733
```

```
# On average we are off by 2404 dollars
```

Example 4 – K-fold Cross Validation

```
all_scores = []
```

 # modify the for loop to record MAE values after each epoch

```
for i in range(k):
    print ('processing fold #', i)

    # Select ith fold test set
    X_val = Xtrain_scaled[i*n:(i+1)*n]
    y_val = y_train[i*n:(i+1)*n]

    # Create ith fold train set
    X_fold = np.concatenate([Xtrain_scaled[:i*n],
                             Xtrain_scaled[(i+1)*n:]], axis = 0)
    y_fold = np.concatenate([y_train[:i*n],
                             y_train[(i+1)*n:]],  axis = 0)

    model = build_model()
    # Train ith fold
    output = model.fit(X_fold,y_fold, validation_data=(X_val,y_val),
                       epochs = 100, batch_size = 1,verbose = 0)
    mae_history = output.history['val_mae']
    all_scores.append(mae_history)
```

Example 4 – Tuning the n. of epochs

```
len(all_scores)
```

```
4
```

```
len(all_scores[0])
```

```
100
```

 # a list of 4 lists, each with 100 MAE values

```
array1 = np.vstack(all_scores).T
array1[:5]
```

```
array([[4.0470109 , 5.44161797, 4.56757212, 4.26246643],
       [3.16702819, 3.82390976, 3.25191021, 3.51601124],
       [2.53141928, 3.29371619, 2.86713576, 2.63695264],
       [3.05941558, 3.74093485, 2.81018758, 2.5811286 ],
       [2.25084043, 2.91507649, 2.56233144, 2.67849207]])
```

100 x 4

Example 4 – Tuning the n. of epochs

```
cols = range(1,5)
df2 = pd.DataFrame(array1,columns = cols)
df2.columns.name = 'fold'
df2.index.name = 'epoch'
df2
```

fold	1	2	3	4
epoch				
0	4.047011	5.441618	4.567572	4.262466
1	3.167028	3.823910	3.251910	3.516011
2	2.531419	3.293716	2.867136	2.636953
3	3.059416	3.740935	2.810188	2.581129
4	2.250840	2.915076	2.562331	2.678492
...	...	...	...	...
95	2.432788	2.733259	2.579102	2.264670
96	2.847583	2.804327	2.272040	2.197043
97	3.012051	2.857696	2.186508	2.146194
98	2.552628	2.961175	2.283123	2.696445
99	3.035638	2.684096	2.433445	2.279399

```
df3 = df2.copy()
df3['means'] = df3.mean(axis=1)
df3[:5]
```

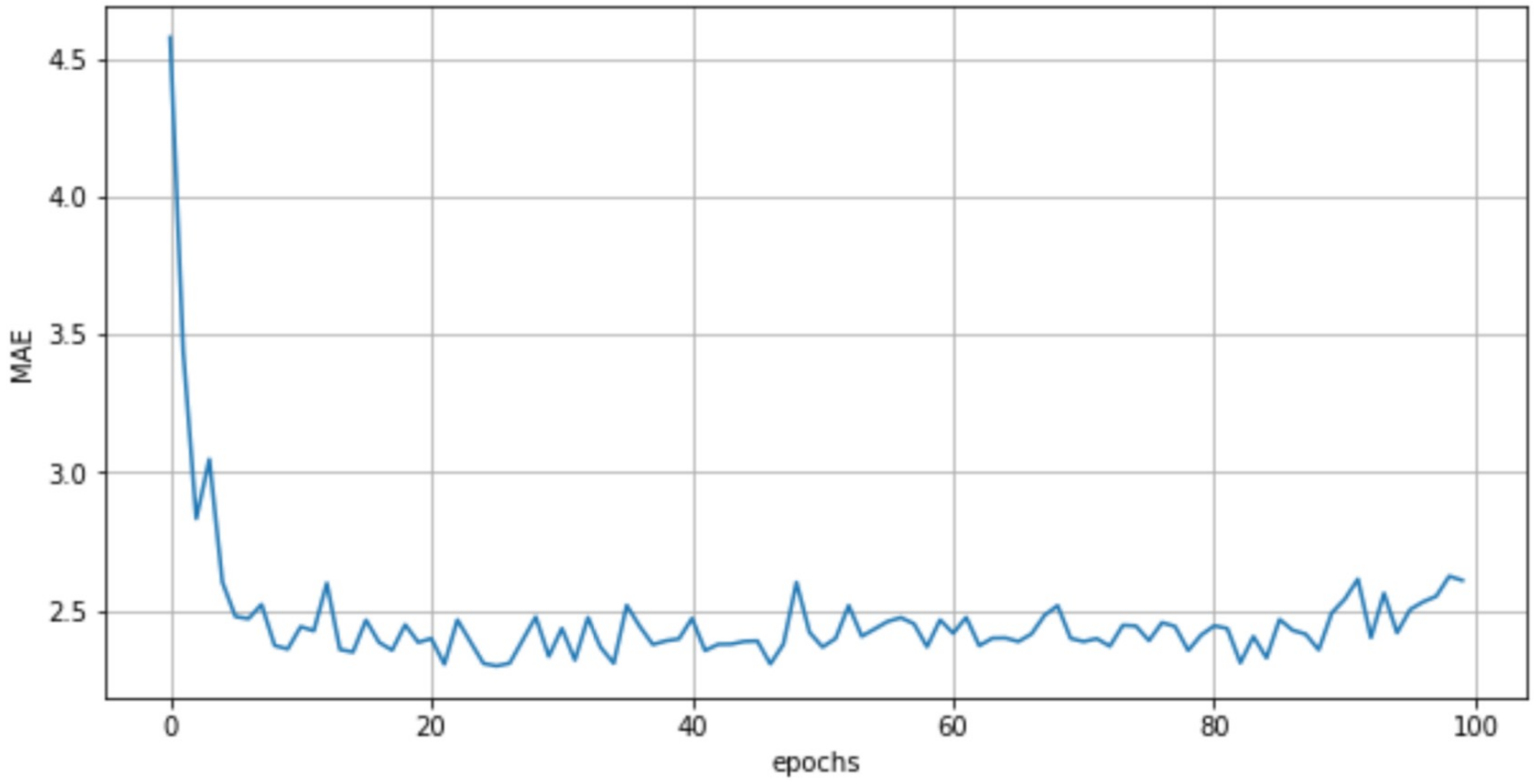
row

fold	1	2	3	4	means
epoch					
0	4.047011	5.441618	4.567572	4.262466	4.579667
1	3.167028	3.823910	3.251910	3.516011	3.439715
2	2.531419	3.293716	2.867136	2.636953	2.832306
3	3.059416	3.740935	2.810188	2.581129	3.047917
4	2.250840	2.915076	2.562331	2.678492	2.601685

```
means = df3.means
```

Example 4 – Tuning the n. of epochs

```
xaxis = range(100)
plt.figure(figsize =(10,5))
plt.plot(xaxis,means)
plt.xlabel('epochs')
plt.ylabel('MAE')
```



Example 4 – Tuning the n. of epochs

```
means = df3.means
means
```

```
epoch
0      4.579667
1      3.439715
2      2.832306
3      3.047917
4      2.601685
          ...
95     2.502455
96     2.530248
97     2.550612
98     2.623343
99     2.608145
```

```
mavg[mavg == mavg.min()]
```

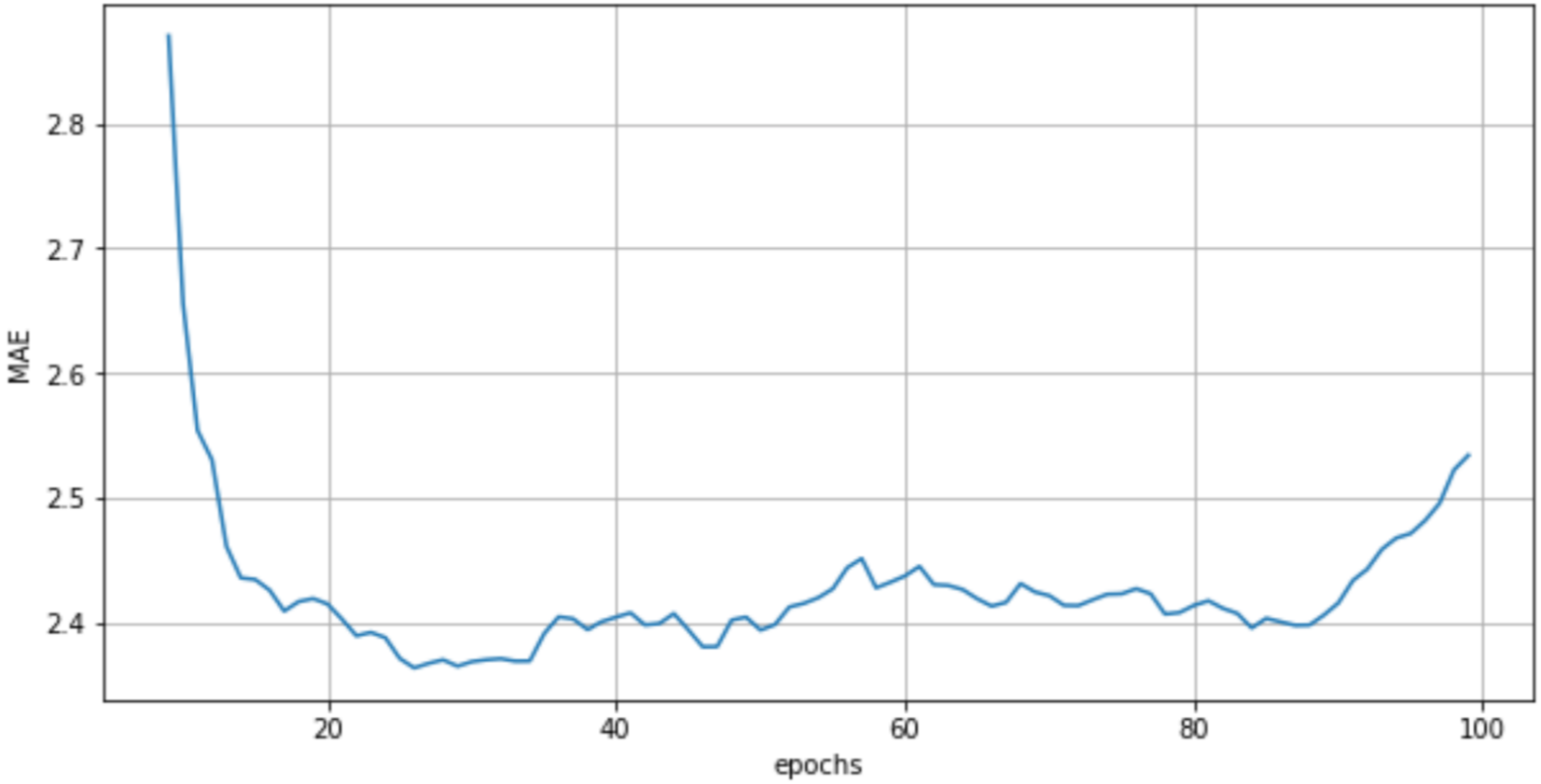
```
epoch
26     2.364716
```

smallest average MAE
found in epoch 26

Example 4 – Tuning the n. of epochs

```
plt.figure(figsize =(10,5))
plt.plot(xaxis,mavg)
plt.xlabel('epochs')
plt.ylabel('MAE')
plt.grid()
```

NN starts overfitting after 26 epochs



Example 4 – Tuning the n. of epochs

```
model = Sequential()
model.add(Dense(64,activation = 'relu', input_shape=(13,)))
model.add(Dense(64,activation = 'relu'))
model.add(Dense(1))
model.compile(optimizer = 'rmsprop', loss = 'mse', metrics=['mae'])
model.fit(Xtrain_scaled,y_train,epochs =26, batch_size = 1,verbose = 0);
```

```
mse, mae = model.evaluate(Xtest_scaled,y_test)
print(mse,',',mae)
```

```
102/102 [==============================] - 0s 344us/step
22.878724850860298 , 2.7827768325805664
```

```
# predictions are off by about 2800 dollars
```