In [2]:

```
from keras.datasets import boston_housing
import numpy as np
from keras import models
from keras import layers
from keras import optimizers
from keras import losses
from keras import metrics
import matplotlib.pyplot as plt

(train_data, train_targets), (test_data, test_targets) = boston_housing.load_d
ata()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-data
sets/boston_housing.npz
57344/57026 [==============================] - 0s 1us/step
```

In [3]:

```
train_data.shape
```

Out[3]:

```
(404, 13)
```

In [4]:

```
mean = train_data.mean(axis = 0)
train_data -= mean
std = train_data.std(axis = 0)
train_data /= std

test_data -= mean
test_data /= std
```

In [6]:

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

In [13]:

```
k = 4
num_val_samples = len(train_data) // k
num_epochs = 100
all_scores = []

for i in range(k):
    print('processing fold #', i)
    val_data = train_data[i * num_val_samples: (i + 1) * num_val_samples]
    val_targets = train_targets[i * num_val_samples: (i + 1) * num_val_samples
]

    partial_train_data = np.concatenate(
        [train_data[:i * num_val_samples],
        train_data[(i + 1) * num_val_samples:]],
        axis = 0)

    partial_train_targets = np.concatenate(
        [train_targets[:i * num_val_samples],
        train_targets[(i + 1) * num_val_samples:]],
        axis = 0)
    model = build_model()
    model.fit(partial_train_data, partial_train_targets, epochs = num_epochs,
batch_size = 1, verbose = 0)
    val_mse, val_mae = model.evaluate(val_data, val_targets, verbose = 0)
    all_scores.append(val_mae)
```

```
processing fold # 0
processing fold # 1
processing fold # 2
processing fold # 3
```

Out[13]:

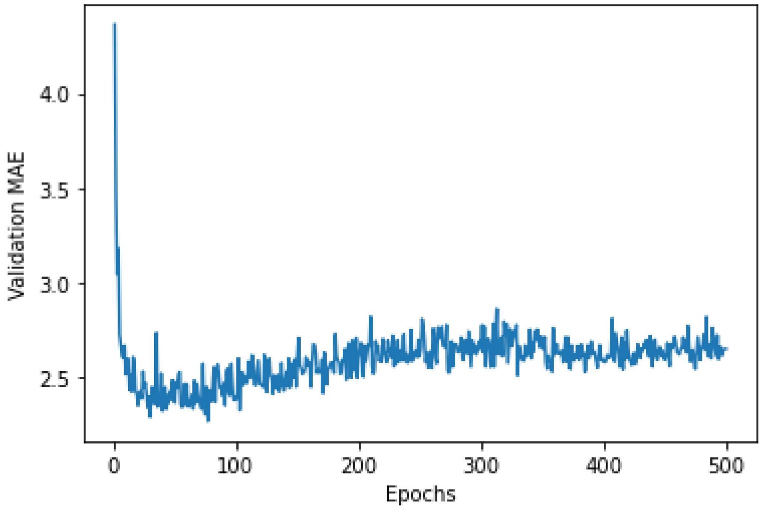
```
" \n    history = model.fit(partial_train_data, partial_train_targets,\n
validation_data = (val_data, val_targets),\n                       epochs = n
um_epochs, batch_size = 1, verbose = 0)\n    mae_history = history.history['v
al_mae']\n    all_mae_histories.append(mae_history)\n"
```

```
In [14]: num_epochs = 500
         all_mae_histories = []

         for i in range(k):
             print('processing fold #', i)
             val_data = train_data[i * num_val_samples: (i + 1) * num_val_samples]
             val_targets = train_targets[i * num_val_samples: (i + 1) * num_val_samples
         ]

             partial_train_data = np.concatenate(
                 [train_data[:i * num_val_samples],
                 train_data[(i + 1) * num_val_samples:]],
                 axis = 0)

             partial_train_targets = np.concatenate(
                 [train_targets[:i * num_val_samples],
                 train_targets[(i + 1) * num_val_samples:]],
                 axis = 0)
             model = build_model()
             history = model.fit(partial_train_data, partial_train_targets,
                                 validation_data = (val_data, val_targets),
                                 epochs = num_epochs, batch_size = 1, verbose = 0)
             mae_history = history.history['val_mae']
             all_mae_histories.append(mae_history)
```

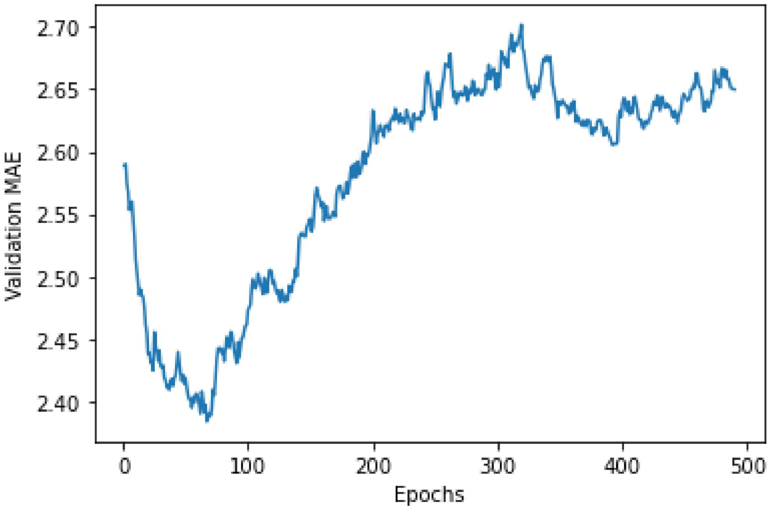
```
processing fold # 0
processing fold # 1
processing fold # 2
processing fold # 3
```

```
In [16]: average_mae_history = [np.mean([x[i] for x in all_mae_histories]) for i in ran
         ge(num_epochs)]
```

```
In [25]: plt.plot(range(1, len(average_mae_history) + 1), average_mae_history)
         plt.xlabel('Epochs')
         plt.ylabel('Validation MAE')
         plt.show()
```



In [29]:
```
def smooth_curve(points, factor = 0.9):
    smoothed_points = []
    for point in points:
        if smoothed_points:
            previous = smoothed_points[-1]
            smoothed_points.append(previous * factor + point * (1 - factor))
        else:
            smoothed_points.append(point)
    return smoothed_points

smooth_mae_history = smooth_curve(average_mae_history[10:])

plt.plot(range(1, len(smooth_mae_history) + 1), smooth_mae_history)
plt.xlabel('Epochs')
plt.ylabel('Validation MAE')
plt.show()
```



In [21]:
```
model = build_model()
model.fit(train_data, train_targets, epochs = 80, batch_size = 16, verbose = 0
)
test_mse_score, test_mae_score = model.evaluate(test_data, test_targets)
```

```
4/4 [==============================] - 0s 1ms/step - loss: 19.0147 - mae: 2.6
563
```

In [23]:
```
smooth_mae_history
```

Out[23]:
```
[2.589105248451233]
```

In [ ]: