Assignment 5.3 - Predicting house price

Loading the dataset

In [1]:
```
import numpy as np
from tensorflow.keras.datasets import boston_housing
```

In [3]:
```
# train and test set
(train_data, train_targets), (test_data, test_targets) = boston_housing.l
```

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

In [4]:
```
train_data.shape
```

Out[4]: (404, 13)

In [5]:
```
test_data.shape
```

Out[5]: (102, 13)

In [9]:
```
train_targets[:10]
```

Out[9]: array([15.2, 42.3, 50. , 21.1, 17.7, 18.5, 11.3, 15.6, 15.6, 14.4])

Preparing the data

In [10]:
```
# Normalizing the data
mean = train_data.mean(axis = 0)
train_data -= mean

std = train_data.std(axis = 0)
train_data /= std

test_data -= mean
test_data /= std
```

Building the network

In [11]:
```
# Import keras libraries
from keras import models, layers, losses, metrics, optimizers
```

```
Using TensorFlow backend.
```

In [12]:
```
# The network ends with a single unit and no activation (it will be a lin
def build_model():
    model = models.Sequential()
    model.add(layers.Dense(64, activation = 'relu', input_shape = (train_
    model.add(layers.Dense(64, activation = 'relu'))
    model.add(layers.Dense(1))
    model.compile(optimizer = 'rmsprop', loss = 'mse', metrics = ['mae']
    return model
```

Validating the model

In [13]:
```
# cross Validation
k = 4
num_val_samples = len(train_data) // k
num_epochs = 100
all_scores = []

for i in range(k):
    print(f'Processing Fold #{i+1}')
    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_sa

    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=False)

    val_mse, val_mae = model.evaluate(val_data, val_targets, verbose=Fals

    all_scores.append(val_mae)
```

```
Processing Fold #1
Processing Fold #2
Processing Fold #3
Processing Fold #4
```

In [14]:
```
all_scores
```

Out[14]:
```
[2.0902836322784424,
 2.2470052242279053,
 2.8744683265686035,
 2.4338009357452393]
```

In [15]:
```
np.mean(all_scores)
```

Out[15]: 2.4113895297050476

In [16]:
```
# training the network with 500 epochs
num_epochs = 500
all_mae_histories = []
for i in range(k):
    print(f'Processing Fold #{i+1}')
    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_sa

    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 #1
Processing Fold #2
Processing Fold #3
Processing Fold #4
```

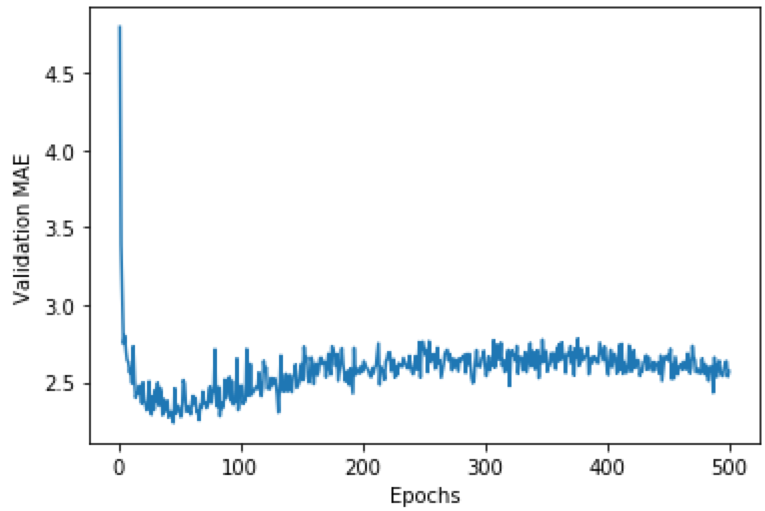
In [17]:
```
average_mae_history = [np.mean([x[i] for x in all_mae_histories]) for i
```

Plotting the training and validation loss and accuracy

In [18]:
```
import matplotlib.pyplot as plt
```

In [19]:

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

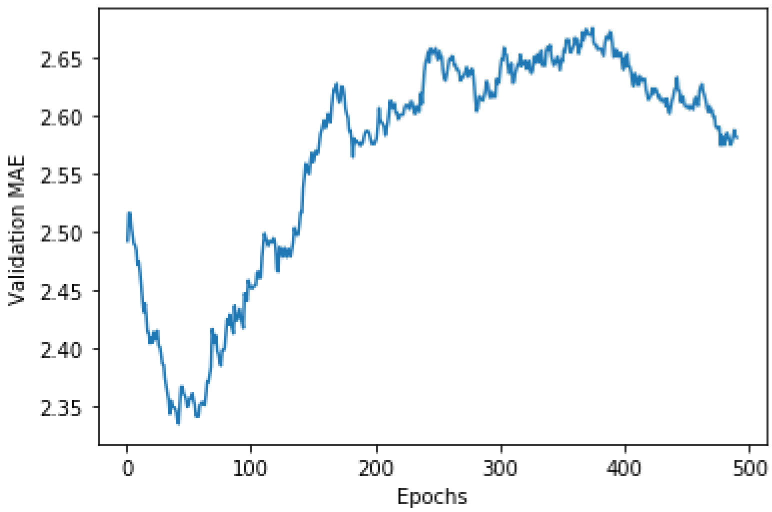


In [20]:
```
# plotting validation scores excluding the first 10 data points
def smooth_curve(points, factor=0.9):
    smoothed_points = []
    for point in points:
        if smoothed_points:
            previous = smoothed_points[-1]
            smoothed_points.append(previous * factor + point * (1 - fact
        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()

```



Training the final model

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)
```

```
102/102 [==============================] - 0s 165us/step
```

In [22]:
```
test_mae_score
```

Out[22]:
```
2.6750268936157227
```