Linear Regression using TensorFlow

Import the necessary libraries

```
from sklearn.datasets import fetch_california_housing
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import tensorflow as tf
from tensorflow import keras
from sklearn.metrics import accuracy_score
```

Fetch the data (here california housing dataset is used)

```
housing = fetch_california_housing()
housing
```

```
{'data': array([[   8.3252    ,   41.        ,    6.98412698, ...,
2.55555556,
           37.88      , -122.23      ],
        [   8.3014    ,   21.        ,    6.23813708, ...,
2.10984183,
           37.86      , -122.22      ],
        [   7.2574    ,   52.        ,    8.28813559, ...,
2.80225989,
           37.85      , -122.24      ],
        ...,
        [   1.7       ,   17.        ,    5.20554273, ...,
2.3256351 ,
           39.43      , -121.22      ],
        [   1.8672    ,   18.        ,    5.32951289, ...,
2.12320917,
           39.43      , -121.32      ],
        [   2.3886    ,   16.        ,    5.25471698, ...,
2.61698113,
           39.37      , -121.24      ]]),
 'target': array([4.526, 3.585, 3.521, ..., 0.923, 0.847, 0.894]),
 'frame': None,
 'target_names': ['MedHouseVal'],
 'feature_names': ['MedInc',
  'HouseAge',
  'AveRooms',
  'AveBedrms',
  'Population',
  'AveOccup',
  'Latitude',
  'Longitude'],
 'DESCR': '.. _california_housing_dataset:\n\nCalifornia Housing
```

```
dataset\n---------------------------\n\n**Data Set Characteristics:**\
n\n    :Number of Instances: 20640\n\n    :Number of Attributes: 8
numeric, predictive attributes and the target\n\n    :Attribute
Information:\n        - MedInc        median income in block group\n
- HouseAge      median house age in block group\n        - AveRooms
average number of rooms per household\n        - AveBedrms     average
number of bedrooms per household\n        - Population    block group
population\n        - AveOccup      average number of household
members\n        - Latitude      block group latitude\n        -
Longitude     block group longitude\n\n    :Missing Attribute Values:
None\n\nThis dataset was obtained from the StatLib repository.\
nhttps://www.dcc.fc.up.pt/~ltorgo/Regression/cal_housing.html\n\nThe
target variable is the median house value for California districts,\
nexpressed in hundreds of thousands of dollars ($100,000).\n\nThis
dataset was derived from the 1990 U.S. census, using one row per
census\nblock group. A block group is the smallest geographical unit
for which the U.S.\nCensus Bureau publishes sample data (a block group
typically has a population\nof 600 to 3,000 people).\n\nA household is
a group of people residing within a home. Since the average\nnumber of
rooms and bedrooms in this dataset are provided per household, these\
ncolumns may take surprisingly large values for block groups with few
households\nand many empty houses, such as vacation resorts.\n\nIt can
be downloaded/loaded using the\
n:func:`sklearn.datasets.fetch_california_housing` function.\n\n..
topic:: References\n\n    - Pace, R. Kelley and Ronald Barry, Sparse
Spatial Autoregressions,\n      Statistics and Probability Letters, 33
(1997) 291-297\n'}
```

Data Preprocessing

Splitting the full data into train and test

```
X_train_full, X_test, y_train_full, y_test =
train_test_split(housing.data, housing.target)
X_train_full, X_test, y_train_full, y_test
```

```
(array([[   4.6053    ,   30.        ,    4.46795435, ...,
1.69885865,
           37.76      , -122.44      ],
        [   3.3906    ,   33.        ,    5.69555035, ...,
2.68618267,
           32.78      , -116.95      ],
        [   5.2806    ,   47.        ,    4.48533333, ...,    2.232
,
           37.78      , -122.46      ],
        ...,
        [   3.2663    ,   19.        ,    3.604913  , ...,
1.86284545,
           34.04      , -118.46      ],
```

```
        [   7.1572    ,   25.        ,    7.70588235, ...,
3.07969639,
           33.76      , -118.04      ],
        [   3.0965    ,   16.        ,    3.32477169, ...,
1.93493151,
           34.02      , -118.41      ]]),
 array([[   1.7054    ,   39.        ,    3.90082645, ...,
3.38347107,
           37.63      , -120.97      ],
        [   2.6442    ,   34.        ,    6.18592965, ...,
4.22613065,
           34.01      , -117.41      ],
        [   2.3382    ,   19.        ,    4.05989111, ...,
2.60980036,
           37.97      , -122.34      ],
        ...,
        [   4.2554    ,   50.        ,    6.36538462, ...,
2.58461538,
           34.02      , -118.32      ],
        [   5.4284    ,    4.        ,    5.06722689, ...,
1.85714286,
           33.65      , -117.86      ],
        [   3.1964    ,   29.        ,    3.97790055, ...,
5.77348066,
           33.74      , -117.88      ]]),
 array([3.861, 1.381, 4.   , ..., 3.375, 2.949, 3.2  ]),
 array([0.588  , 0.92   , 1.207  , ..., 1.943  , 5.00001, 1.519  ]))
```

Splitting the training data

```
X_train, X_valid, y_train, y_valid = train_test_split(X_train_full,
y_train_full)
X_train, X_valid, y_train, y_valid
```

```
(array([[   4.9375    ,   19.        ,    4.61764706, ...,    2.34375
,
           33.87      , -118.36      ],
        [   6.2045    ,   29.        ,    5.95192308, ...,
2.92548077,
           37.31      , -122.04      ],
        [   6.1943    ,   13.        ,    6.53246753, ...,
3.46753247,
           38.68      , -121.25      ],
        ...,
        [   3.1429    ,   32.        ,    5.98951049, ...,
3.11538462,
           33.11      , -117.09      ],
        [   3.625     ,   20.        ,    3.56113903, ...,
1.86264657,
           37.57      , -122.33      ],
```

```
        [   3.5917    ,   22.        ,    5.41052632, ...,
2.88070175,
           39.21      , -123.19      ]]),
 array([[   2.375     ,   11.        ,    5.96511628, ...,
2.37209302,
           39.78      , -120.48      ],
        [   3.9479    ,   35.        ,    5.36286201, ...,
3.59454855,
           33.78      , -117.97      ],
        [   2.4387    ,   34.        ,    2.74399038, ...,
3.64423077,
           33.98      , -118.22      ],
        ...,
        [   5.3054    ,   36.        ,    5.86038961, ...,
2.86363636,
           33.96      , -118.01      ],
        [   7.2423    ,   52.        ,    7.875     , ...,    5.875
,
           33.89      , -118.17      ],
        [   2.9327    ,   42.        ,    4.72705314, ...,
2.37922705,
           34.18      , -118.39      ]]),
 array([2.93 , 3.938, 2.761, ..., 1.716, 2.833, 1.188]),
 array([1.   , 2.038, 1.75 , ..., 2.735, 3.5  , 2.402]))
```

Dimensionality Reduction using transform and fit transform

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

```
array([[ 0.55699857, -0.77113174, -0.33874061, ..., -0.05915181,
        -0.81896681,  0.59911607],
       [ 1.22012201,  0.02706727,  0.21292832, ..., -0.01652659,
         0.78841092, -1.23817849],
       [ 1.21478353, -1.25005115,  0.45295987, ...,  0.02319122,
         1.42855845, -0.84375928],
       ...,
       [-0.38226056,  0.26652697,  0.22846919, ..., -0.00261174,
        -1.17408515,  1.2331824 ],
       [-0.12993869, -0.69131184, -0.77556376, ..., -0.09440375,
         0.90989878, -1.38296529],
       [-0.14736727, -0.53167204, -0.01091724, ..., -0.01980769,
         1.67620677, -1.81233304]])
```

```
X_valid = scaler.transform(X_valid)
X_valid
```

```
array([[-0.78416465, -1.40969095,  0.21838318, ..., -0.05707503,
         1.94254552, -0.45932536],
       [ 0.03906096,  0.50598667, -0.0306245 , ...,  0.03249809,
        -0.8610203 ,  0.79382936],
       [-0.7508253 ,  0.42616677, -1.11342152, ...,  0.03613846,
        -0.76756811,  0.66901315],
       ...,
       [ 0.74955036,  0.58580657,  0.17508295, ..., -0.02105812,
        -0.77691333,  0.77385876],
       [ 1.76328658,  1.86292498,  1.0080425 , ...,  0.19959384,
        -0.80962159,  0.69397639],
       [-0.4922752 ,  1.06472598, -0.29350564, ..., -0.05655229,
        -0.67411591,  0.58413813]])
```

```
X_test = scaler.transform(X_test)
X_test
```

```
array([[-1.13462042,  0.82526627, -0.63511681, ...,  0.01703178,
         0.93793443, -0.70396513],
       [-0.64327055,  0.42616677,  0.30968053, ...,  0.07877607,
        -0.75355028,  1.07341766],
       [-0.80342507, -0.77113174, -0.56935003, ..., -0.03965747,
         1.09680316, -1.38795794],
       ...,
       [ 0.20000055,  1.70328518,  0.38387787, ..., -0.04150285,
        -0.74887767,  0.61908667],
       [ 0.81392619, -1.96843025, -0.15285761, ..., -0.09480702,
        -0.92176423,  0.84874849],
       [-0.35425969,  0.02706727, -0.6032498 , ...,  0.1921552 ,
        -0.87971074,  0.83876319]])
```

Training

Selecting the model for neural network with the activation function

```
model = keras.models.Sequential([keras.layers.Dense(30,
activation="relu", input_shape=X_train.shape[1:]),
keras.layers.Dense(1)])
model.compile(loss="mean_squared_error", optimizer="sgd")
```

Training the model

```
history = model.fit(X_train, y_train, epochs=20,
validation_data=(X_valid, y_valid))
```

```
Epoch 1/20
363/363 ━━━━━━━━━━━━━━━━━━━━ 0s 830us/step - loss: 1.3878 - val_loss:
0.5445
```

```
Epoch 2/20
363/363 ━━━━━━━━━━━━━━━━━━━━ 0s 600us/step - loss: 0.5569 - val_loss:
0.4794
Epoch 3/20
363/363 ━━━━━━━━━━━━━━━━━━━━ 0s 597us/step - loss: 0.4879 - val_loss:
0.4456
Epoch 4/20
363/363 ━━━━━━━━━━━━━━━━━━━━ 0s 598us/step - loss: 0.4701 - val_loss:
0.4403
Epoch 5/20
363/363 ━━━━━━━━━━━━━━━━━━━━ 0s 595us/step - loss: 0.4488 - val_loss:
0.4208
Epoch 6/20
363/363 ━━━━━━━━━━━━━━━━━━━━ 0s 639us/step - loss: 0.4372 - val_loss:
0.4117
Epoch 7/20
363/363 ━━━━━━━━━━━━━━━━━━━━ 0s 594us/step - loss: 0.4382 - val_loss:
0.4018
Epoch 8/20
363/363 ━━━━━━━━━━━━━━━━━━━━ 0s 598us/step - loss: 0.4230 - val_loss:
0.3955
Epoch 9/20
363/363 ━━━━━━━━━━━━━━━━━━━━ 0s 650us/step - loss: 0.4100 - val_loss:
0.3954
Epoch 10/20
363/363 ━━━━━━━━━━━━━━━━━━━━ 0s 632us/step - loss: 0.4232 - val_loss:
0.3888
Epoch 11/20
363/363 ━━━━━━━━━━━━━━━━━━━━ 0s 643us/step - loss: 0.4228 - val_loss:
0.3866
Epoch 12/20
363/363 ━━━━━━━━━━━━━━━━━━━━ 0s 591us/step - loss: 0.4048 - val_loss:
0.3807
Epoch 13/20
363/363 ━━━━━━━━━━━━━━━━━━━━ 0s 598us/step - loss: 0.3904 - val_loss:
0.5836
Epoch 14/20
363/363 ━━━━━━━━━━━━━━━━━━━━ 0s 597us/step - loss: 0.6532 - val_loss:
0.3780
Epoch 15/20
363/363 ━━━━━━━━━━━━━━━━━━━━ 0s 597us/step - loss: 0.4036 - val_loss:
0.3711
Epoch 16/20
363/363 ━━━━━━━━━━━━━━━━━━━━ 0s 597us/step - loss: 0.4080 - val_loss:
0.4029
Epoch 17/20
363/363 ━━━━━━━━━━━━━━━━━━━━ 0s 601us/step - loss: 0.3771 - val_loss:
0.3691
Epoch 18/20
```

```
363/363 ━━━━━━━━━━━━━━━━━━━━ 0s 596us/step - loss: 0.3912 - val_loss:
0.3725
Epoch 19/20
363/363 ━━━━━━━━━━━━━━━━━━━━ 0s 596us/step - loss: 0.3950 - val_loss:
0.3634
Epoch 20/20
363/363 ━━━━━━━━━━━━━━━━━━━━ 0s 602us/step - loss: 0.3773 - val_loss:
0.3607
```

Evaluation

Mean Square Error test

```
mse_test = model.evaluate(X_test, y_test)
mse_test

162/162 ━━━━━━━━━━━━━━━━━━━━ 0s 392us/step - loss: 0.3805

0.39877045154571533
```

Testing for new instances

```
X_new = X_test[:3] # pretend these are new instances
X_new

array([[-1.13462042,  0.82526627, -0.63511681, -0.19139976,  0.5645559
,
         0.01703178,  0.93793443, -0.70396513],
       [-0.64327055,  0.42616677,  0.30968053, -0.0231091 , -
0.52968304,
         0.07877607, -0.75355028,  1.07341766],
       [-0.80342507, -0.77113174, -0.56935003, -0.09031679,
0.01199245,
        -0.03965747,  1.09680316, -1.38795794]])
```

Predicting for new instances

```
y_pred = model.predict(X_new)
y_pred

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 33ms/step

array([[0.8186285 ],
       [0.82131505],
       [1.6618688 ]], dtype=float32)
```

Finding the loss

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

```
pd.DataFrame(history.history).plot(figsize=(8, 5))
plt.grid(True)
plt.gca().set_ylim(0, 1)
plt.show()
```