dl-practical-1

April 30, 2024

Linear regression by using Deep Neural network: Implement Boston housing price prediction problem by Linear regression using Deep Neural network. Use Boston House price prediction dataset.

[1]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.datasets import fetch_openml

# Load dataset from UCI repository
data = fetch_openml(data_id=531)

# Convert to pandas DataFrame
df = pd.DataFrame(data.data, columns=data.feature_names)
df['target'] = data.target
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import tensorflow as tf
from sklearn.metrics import accuracy_score
```

```
C:\ProgramData\anaconda3\Lib\site-packages\sklearn\datasets\_openml.py:968:
FutureWarning: The default value of `parser` will change from `'liac-arff'` to
`'auto'` in 1.4. You can set `parser='auto'` to silence this warning. Therefore,
an `ImportError` will be raised from 1.4 if the dataset is dense and pandas is
not installed. Note that the pandas parser may return different data types. See
the Notes Section in fetch_openml's API doc for details.
  warn(
```

[2]:
```
data = fetch_openml(data_id=531)

# Convert to pandas DataFrame
df = pd.DataFrame(data.data, columns=data.feature_names)
df['PRICE'] = data.target

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(df.iloc[:, :-1], df.iloc[:,␣
 ↪-1], test_size=0.2, random_state=42)
```

```
C:\ProgramData\anaconda3\Lib\site-packages\sklearn\datasets\_openml.py:968:
FutureWarning: The default value of `parser` will change from `'liac-arff'` to
`'auto'` in 1.4. You can set `parser='auto'` to silence this warning. Therefore,
an `ImportError` will be raised from 1.4 if the dataset is dense and pandas is
not installed. Note that the pandas parser may return different data types. See
the Notes Section in fetch_openml's API doc for details.
  warn(
```

[3]:
```
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

[4]:
```
model = tf.keras.models.Sequential([
tf.keras.layers.Dense(64, activation='relu', input_shape=(X_train.shape[1],)),
tf.keras.layers.Dense(32, activation='relu'),
tf.keras.layers.Dense(1)
])
# Compile the model
model.compile(optimizer='adam', loss='mse')
2
 # Train the model
history = model.fit(X_train, y_train, epochs=100,␣
 ↪batch_size=32,validation_split=0.2)
```

```
C:\ProgramData\anaconda3\Lib\site-packages\keras\src\layers\core\dense.py:86:
UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When
using Sequential models, prefer using an `Input(shape)` object as the first
layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)

Epoch 1/100
11/11             3s 35ms/step -
loss: 643.9668 - val_loss: 538.5846
Epoch 2/100
11/11             0s 7ms/step - loss:
589.4406 - val_loss: 507.0188
Epoch 3/100
11/11             0s 6ms/step - loss:
537.2333 - val_loss: 470.9958
Epoch 4/100
11/11             0s 6ms/step - loss:
462.0255 - val_loss: 426.6866
Epoch 5/100
11/11             0s 7ms/step - loss:
489.3096 - val_loss: 371.9767
Epoch 6/100
11/11             0s 8ms/step - loss:
399.1812 - val_loss: 307.2765
```

```
Epoch 7/100
11/11             0s 9ms/step - loss:
296.7974 - val_loss: 236.9038
Epoch 8/100
11/11             0s 6ms/step - loss:
244.3759 - val_loss: 169.5194
Epoch 9/100
11/11             0s 6ms/step - loss:
162.3032 - val_loss: 117.0605
Epoch 10/100
11/11             0s 9ms/step - loss:
129.1309 - val_loss: 81.1631
Epoch 11/100
11/11             0s 6ms/step - loss:
93.0238 - val_loss: 60.5970
Epoch 12/100
11/11             0s 6ms/step - loss:
79.5418 - val_loss: 47.9206
Epoch 13/100
11/11             0s 6ms/step - loss:
61.3981 - val_loss: 40.5878
Epoch 14/100
11/11             0s 9ms/step - loss:
47.9771 - val_loss: 36.3062
Epoch 15/100
11/11             0s 9ms/step - loss:
43.7164 - val_loss: 34.5451
Epoch 16/100
11/11             0s 9ms/step - loss:
40.3259 - val_loss: 32.5833
Epoch 17/100
11/11             0s 14ms/step -
loss: 29.5888 - val_loss: 31.5790
Epoch 18/100
11/11             0s 7ms/step - loss:
34.6715 - val_loss: 30.5626
Epoch 19/100
11/11             0s 10ms/step -
loss: 26.4467 - val_loss: 30.1054
Epoch 20/100
11/11             0s 10ms/step -
loss: 31.6181 - val_loss: 29.5193
Epoch 21/100
11/11             0s 6ms/step - loss:
27.4579 - val_loss: 29.0490
Epoch 22/100
11/11             0s 6ms/step - loss:
25.3876 - val_loss: 28.4564
```

```
Epoch 23/100
11/11             0s 7ms/step - loss:
24.9924 - val_loss: 27.2889
Epoch 24/100
11/11             0s 8ms/step - loss:
21.9076 - val_loss: 26.5518
Epoch 25/100
11/11             0s 9ms/step - loss:
19.5363 - val_loss: 26.4127
Epoch 26/100
11/11             0s 10ms/step -
loss: 21.2320 - val_loss: 26.3215
Epoch 27/100
11/11             0s 6ms/step - loss:
21.1608 - val_loss: 25.9699
Epoch 28/100
11/11             0s 7ms/step - loss:
17.3395 - val_loss: 25.6190
Epoch 29/100
11/11             0s 6ms/step - loss:
20.2930 - val_loss: 24.9270
Epoch 30/100
11/11             0s 13ms/step -
loss: 17.3946 - val_loss: 24.6827
Epoch 31/100
11/11             0s 8ms/step - loss:
18.3738 - val_loss: 24.6548
Epoch 32/100
11/11             0s 10ms/step -
loss: 17.7488 - val_loss: 24.4575
Epoch 33/100
11/11             0s 10ms/step -
loss: 16.3322 - val_loss: 23.9353
Epoch 34/100
11/11             0s 6ms/step - loss:
17.2695 - val_loss: 23.3180
Epoch 35/100
11/11             0s 6ms/step - loss:
20.8005 - val_loss: 22.7801
Epoch 36/100
11/11             0s 7ms/step - loss:
17.0443 - val_loss: 22.6605
Epoch 37/100
11/11             0s 6ms/step - loss:
16.8359 - val_loss: 22.1736
Epoch 38/100
11/11             0s 6ms/step - loss:
15.0916 - val_loss: 21.5442
```

```
Epoch 39/100
11/11             0s 9ms/step - loss:
14.1819 - val_loss: 21.5228
Epoch 40/100
11/11             0s 11ms/step -
loss: 15.1891 - val_loss: 21.6215
Epoch 41/100
11/11             0s 19ms/step -
loss: 16.3675 - val_loss: 20.9897
Epoch 42/100
11/11             0s 7ms/step - loss:
16.5234 - val_loss: 20.3902
Epoch 43/100
11/11             0s 10ms/step -
loss: 12.5086 - val_loss: 20.2358
Epoch 44/100
11/11             0s 13ms/step -
loss: 13.3497 - val_loss: 20.2145
Epoch 45/100
11/11             0s 11ms/step -
loss: 14.5418 - val_loss: 19.9243
Epoch 46/100
11/11             0s 9ms/step - loss:
12.2214 - val_loss: 19.0724
Epoch 47/100
11/11             0s 7ms/step - loss:
12.6090 - val_loss: 19.1420
Epoch 48/100
11/11             0s 7ms/step - loss:
13.7728 - val_loss: 19.0007
Epoch 49/100
11/11             0s 8ms/step - loss:
13.1584 - val_loss: 18.9081
Epoch 50/100
11/11             0s 8ms/step - loss:
11.7631 - val_loss: 18.7219
Epoch 51/100
11/11             0s 8ms/step - loss:
12.5710 - val_loss: 18.4496
Epoch 52/100
11/11             0s 11ms/step -
loss: 12.2622 - val_loss: 18.3753
Epoch 53/100
11/11             0s 9ms/step - loss:
12.8655 - val_loss: 18.0539
Epoch 54/100
11/11             0s 5ms/step - loss:
13.0847 - val_loss: 17.6926
```

```
Epoch 55/100
11/11             0s 7ms/step - loss:
13.0746 - val_loss: 17.1167
Epoch 56/100
11/11             0s 8ms/step - loss:
13.4907 - val_loss: 17.0395
Epoch 57/100
11/11             0s 11ms/step -
loss: 12.6376 - val_loss: 17.0174
Epoch 58/100
11/11             0s 8ms/step - loss:
11.4287 - val_loss: 17.3412
Epoch 59/100
11/11             0s 9ms/step - loss:
11.4611 - val_loss: 16.9692
Epoch 60/100
11/11             0s 7ms/step - loss:
11.3522 - val_loss: 18.1204
Epoch 61/100
11/11             0s 9ms/step - loss:
11.8720 - val_loss: 18.1520
Epoch 62/100
11/11             0s 8ms/step - loss:
10.8296 - val_loss: 18.2964
Epoch 63/100
11/11             0s 7ms/step - loss:
11.1775 - val_loss: 17.7540
Epoch 64/100
11/11             0s 23ms/step -
loss: 10.1066 - val_loss: 17.3420
Epoch 65/100
11/11             0s 10ms/step -
loss: 13.3626 - val_loss: 16.9325
Epoch 66/100
11/11             0s 10ms/step -
loss: 9.6593 - val_loss: 16.9357
Epoch 67/100
11/11             0s 12ms/step -
loss: 12.0105 - val_loss: 16.4199
Epoch 68/100
11/11             0s 8ms/step - loss:
11.7115 - val_loss: 16.2466
Epoch 69/100
11/11             0s 7ms/step - loss:
10.0814 - val_loss: 15.9863
Epoch 70/100
11/11             0s 14ms/step -
loss: 10.6282 - val_loss: 15.9115
```

```
Epoch 71/100
11/11             0s 8ms/step - loss:
11.0275 - val_loss: 15.9930
Epoch 72/100
11/11             0s 6ms/step - loss:
12.9148 - val_loss: 15.8629
Epoch 73/100
11/11             0s 7ms/step - loss:
10.4210 - val_loss: 15.6122
Epoch 74/100
11/11             0s 7ms/step - loss:
10.4924 - val_loss: 15.8747
Epoch 75/100
11/11             0s 7ms/step - loss:
9.7016 - val_loss: 15.5388
Epoch 76/100
11/11             0s 14ms/step -
loss: 11.5798 - val_loss: 15.7215
Epoch 77/100
11/11             0s 7ms/step - loss:
10.4065 - val_loss: 15.7846
Epoch 78/100
11/11             0s 7ms/step - loss:
11.4981 - val_loss: 16.6097
Epoch 79/100
11/11             0s 7ms/step - loss:
11.6422 - val_loss: 15.9455
Epoch 80/100
11/11             0s 12ms/step -
loss: 9.2596 - val_loss: 15.6749
Epoch 81/100
11/11             0s 7ms/step - loss:
9.8477 - val_loss: 16.0211
Epoch 82/100
11/11             0s 9ms/step - loss:
8.7895 - val_loss: 15.6523
Epoch 83/100
11/11             0s 7ms/step - loss:
11.1307 - val_loss: 15.3257
Epoch 84/100
11/11             0s 6ms/step - loss:
10.1470 - val_loss: 14.9773
Epoch 85/100
11/11             0s 8ms/step - loss:
8.6412 - val_loss: 15.0512
Epoch 86/100
11/11             0s 6ms/step - loss:
9.0321 - val_loss: 14.6110
```

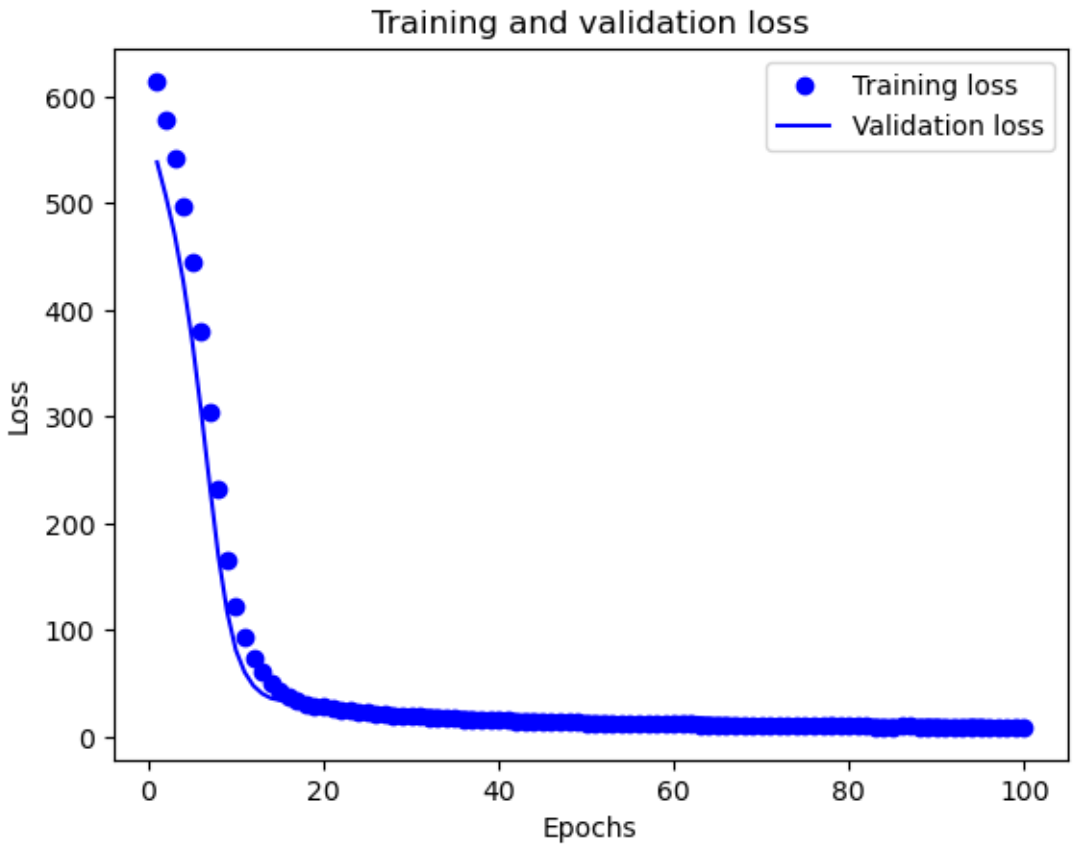
```
Epoch 87/100
11/11             0s 7ms/step - loss:
9.6045 - val_loss: 14.6369
Epoch 88/100
11/11             0s 15ms/step -
loss: 10.3673 - val_loss: 15.1431
Epoch 89/100
11/11             0s 8ms/step - loss:
11.3909 - val_loss: 15.4719
Epoch 90/100
11/11             0s 9ms/step - loss:
10.3872 - val_loss: 15.3994
Epoch 91/100
11/11             0s 7ms/step - loss:
8.7119 - val_loss: 14.7432
Epoch 92/100
11/11             0s 7ms/step - loss:
9.4753 - val_loss: 14.7152
Epoch 93/100
11/11             0s 6ms/step - loss:
8.9547 - val_loss: 14.6985
Epoch 94/100
11/11             0s 17ms/step -
loss: 9.8003 - val_loss: 15.3511
Epoch 95/100
11/11             0s 8ms/step - loss:
9.4212 - val_loss: 15.1819
Epoch 96/100
11/11             0s 7ms/step - loss:
10.0960 - val_loss: 14.4403
Epoch 97/100
11/11             0s 10ms/step -
loss: 7.6341 - val_loss: 14.3136
Epoch 98/100
11/11             0s 7ms/step - loss:
8.4463 - val_loss: 14.2195
Epoch 99/100
11/11             0s 7ms/step - loss:
9.1117 - val_loss: 14.5455
Epoch 100/100
11/11             0s 23ms/step -
loss: 9.2894 - val_loss: 14.3963
```

[5]:
```
# Evaluate the model
model.evaluate(X_test, y_test)
```

```
4/4             0s 3ms/step - loss:
9.1029
```

[5]: 12.356955528259277

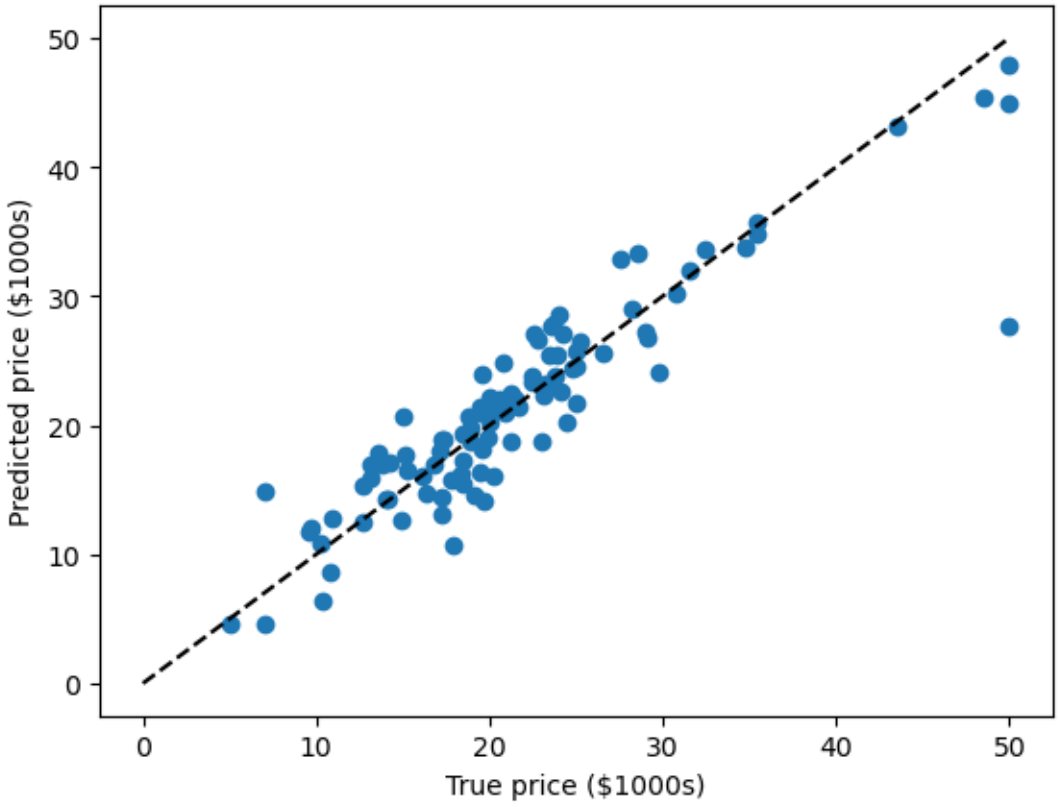
[7]:
```
# Visualize the training history
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(loss) + 1)
plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```



[8]:
```
# Visualize the linear regression graph
y_pred = model.predict(X_test)
plt.scatter(y_test, y_pred)
plt.plot([0, 50], [0, 50], '--k')
plt.axis('tight')
plt.xlabel('True price ($1000s)')
```

```
plt.ylabel('Predicted price ($1000s)')
plt.show()
```

```
4/4             0s 24ms/step
```



[9]:
```
X_train.shape
```

[9]:
```
(404, 13)
```

[10]:
```
X_train[0]
```

[10]:
```
array([ 1.28770177, -0.50032012,  1.03323679, -0.27808871,  0.48925206,
       -1.42806858,  1.02801516, -0.80217296,  1.70689143,  1.57843444,
        0.84534281, -0.07433689,  1.75350503])
```

[11]:
```
custom_input = np.array([[0.2, 0.0, 10.0, 0.0, 0.5, 6.0, 70.0, 3.0, 4.0, 400.
↪0,17.0, 360.0, 15.0]])
scaled_custom_input = scaler.transform(custom_input)
```

```
C:\ProgramData\anaconda3\Lib\site-packages\sklearn\base.py:439: UserWarning: X
does not have valid feature names, but StandardScaler was fitted with feature
```

```
names
  warnings.warn(
```

[12]:
```
custom_prediction = model.predict(scaled_custom_input)
print(f"Custom Input Prediction: {custom_prediction[0][0]}")
```

```
1/1             0s 35ms/step
Custom Input Prediction: 17.20086669921875
```

[ ]:
```

```