

# Assignment 1

```
[2]: import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from sklearn.datasets import fetch_openml
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
```

```
[4]: boston = fetch_openml(name='boston', version=1, as_frame=True)
```

```
X = boston.data
y = boston.target
```

```
# Convert target to numeric
y = y.astype(float)
```

```
print("Features shape:", X.shape)
print("Target shape:", y.shape)
```

```
Features shape: (506, 13)
Target shape: (506,)
```

```
[6]: X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42)
```

```
[6]: X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)
```

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

```
[7]: model = keras.Sequential([
    layers.Dense(64, activation='relu', input_shape=(X_train.shape[1],)),
    layers.Dense(32, activation='relu'),
    layers.Dense(16, activation='relu'),
    layers.Dense(1) # Output layer for regression
])
```

D:\DL\.venv\lib\site-packages\keras\src\layers\core\dense.py:95: 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)

```
[8]: model.compile(
    optimizer='adam',
    loss='mse',
    metrics=['mae']
)
```

WARNING:tensorflow:TensorFlow GPU support is not available on native Windows for TensorFlow >= 2.11. Even if CUDA/cuDNN are installed, GPU will not be used. Please use WSL2 or the TensorFlow-DirectML plugin.

```
[9]: history = model.fit(
      X_train, y_train,
      epochs=100,
      batch_size=16,
      validation_split=0.2,
      verbose=1
    )
```

```
Epoch 92/100
21/21 — 0s 5ms/step - loss: 4.4786 - mae: 1.5819 - val_loss: 10.7528 - val_mae: 2.3184
Epoch 93/100
21/21 — 0s 5ms/step - loss: 4.4256 - mae: 1.5378 - val_loss: 11.4902 - val_mae: 2.4384
Epoch 94/100
21/21 — 0s 5ms/step - loss: 4.2887 - mae: 1.5310 - val_loss: 10.9075 - val_mae: 2.3391
Epoch 95/100
21/21 — 0s 5ms/step - loss: 4.2988 - mae: 1.5186 - val_loss: 11.6304 - val_mae: 2.4400
Epoch 96/100
21/21 — 0s 5ms/step - loss: 4.4956 - mae: 1.5286 - val_loss: 11.1161 - val_mae: 2.3763
Epoch 97/100
21/21 — 0s 6ms/step - loss: 4.5059 - mae: 1.5638 - val_loss: 10.6155 - val_mae: 2.3265
Epoch 98/100
21/21 — 0s 5ms/step - loss: 4.2967 - mae: 1.5325 - val_loss: 11.5779 - val_mae: 2.4214
Epoch 99/100
21/21 — 0s 5ms/step - loss: 4.5689 - mae: 1.5614 - val_loss: 11.5005 - val_mae: 2.4015
Epoch 100/100
21/21 — 0s 6ms/step - loss: 4.3253 - mae: 1.5263 - val_loss: 10.4268 - val_mae: 2.3013
```

```
[10]: loss, mae = model.evaluate(X_test, y_test)
```

```
print("Test Loss (MSE):", loss)
print("Test MAE:", mae)
```

```
print("Test Loss (MSE):", loss)
print("Test MAE:", mae)
```

```
4/4 — 0s 9ms/step - loss: 10.6068 - mae: 2.2021
Test Loss (MSE): 10.606779098510742
Test MAE: 2.202125310897827
```

```
[11]: y_pred = model.predict(X_test)
```

```
# Compare actual vs predicted
comparison = pd.DataFrame({
    "Actual": y_test.values,
    "Predicted": y_pred.flatten()
})

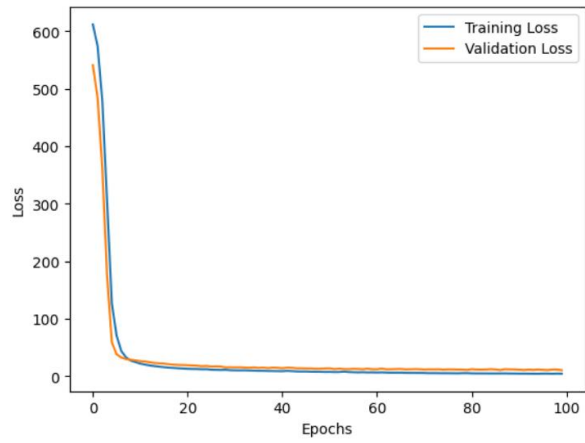
comparison.head()
```

```
4/4 — 0s 19ms/step
```

```
[11]:
```

	Actual	Predicted
0	23.6	27.523327
1	32.4	33.736214
2	13.6	15.979248
3	22.8	24.831005
4	16.1	16.393564

```
[12]: plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```



```
[13]: plt.scatter(y_test, y_pred)
plt.xlabel("Actual Prices")
plt.ylabel("Predicted Prices")
plt.title("Actual vs Predicted")
plt.show()
```

