

# Lab 1: Model Representation (04/09/2024)

Matthew Loh

## Table of contents

<b>1 Imports</b>	<b>1</b>
1.1 Define Training Data . . . . .	2
1.2 Print Training Data Shape and Number of Examples . . . . .	2
<b>2 Plot the Training Data Points</b>	<b>2</b>
2.1 Function to Compute Model Output . . . . .	3
2.2 Function to Plot Model Prediction for Convenience . . . . .	4
<b>3 Control Run</b>	<b>4</b>
3.1 Trial 1 - Increase w, maintain b . . . . .	5
3.2 Trial 2 - Decrease w, maintain b . . . . .	6
3.3 Trial 3 - $w = 200$ , $b = 100$ . . . . .	7
3.4 Trial 4 - Looks Right, not optimum - Increase w by 98, increase b by 2.5 . . . .	8
<b>4 Cost of a 1200 sqft house</b>	<b>9</b>
<b>5 Extra: Calculating Mean Square Error and Jwb Cost Function</b>	<b>9</b>
5.1 Compute Cost for Control Run . . . . .	10
5.2 Compute Cost for Trial 1 . . . . .	10
5.3 Compute Cost for Trial 2 . . . . .	10
5.4 Compute Cost for Trial 3 . . . . .	11
5.5 Compute Cost for Trial 4 . . . . .	11

## 1 Imports

```
import numpy as np
import matplotlib.pyplot as plt
```

## 1.1 Define Training Data

```
# x_train is the input variable (size in 1000 square feet)
# y_train is the target (price in 1000s of dollars)
x_train = np.array([1.0, 2.0])
y_train = np.array([300.0, 500.0])
print(f"x_train: {x_train}")
print(f"y_train: {y_train}")
```

```
x_train: [1. 2.]
y_train: [300. 500.]
```

## 1.2 Print Training Data Shape and Number of Examples

```
print(f"x_train.shape: {x_train.shape}")
m = x_train.shape[0]
print(f"Number of training examples is {m}")

# m is the number of training examples
m = len(x_train)
print(f"Number of training examples is {m}")

i = 0 # Change this to 1 to see (x1, y1)
x_i = x_train[i]
y_i = y_train[i]
print(f"(x{i}), y{i}) = ({x_i}, {y_i})")
```

```
x_train.shape: (2,)
Number of training examples is 2
Number of training examples is 2
(x(0), y(0)) = (1.0, 300.0)
```

## 2 Plot the Training Data Points

```
# Plot the data points
plt.scatter(x_train, y_train, marker='x', c='r')
# Set the title
plt.title("Housing Prices")
# Set the y-axis label
plt.ylabel('Price (in 1000s of dollars)')
# Set the x-axis label
plt.xlabel('Size (1000 sqft)')
plt.show()
```



## 2.1 Function to Compute Model Output

```
def compute_model_output(x: np.ndarray, w: float, b: float) -> np.ndarray:
    """
    Computes the prediction of a linear model
    Args:
        x (np.ndarray (m,)): Data, m examples
        w,b (scalar) : Model parameters
    Returns:
        y (ndarray (m,)) : target values
```

```

"""
m = x.shape[0]
f_wb = np.zeros(m)
for i in range(m):
    f_wb[i] = w * x[i] + b
return f_wb

```

## 2.2 Function to Plot Model Prediction for Convenience

```

def plot_model_prediction(x_train, y_train, w, b):
    """
    Plots the model prediction along with the actual data points
    Args:
        x_train (np.ndarray): Input variable (size in 1000 square feet)
        y_train (np.ndarray): Target variable (price in 1000s of dollars)
        w (float): Model parameter
        b (float): Model parameter
    """
    # Compute the model prediction
    f_wb = compute_model_output(x_train, w, b)

    plt.plot(x_train, f_wb, label='Model Prediction')
    plt.scatter(x_train, y_train, marker='x', c='r', label="Actual Values")
    plt.title("Housing Prices")
    plt.ylabel('Price (in 1000s of dollars)')
    plt.xlabel('Size (1000 sqft)')
    plt.legend()
    plt.show()

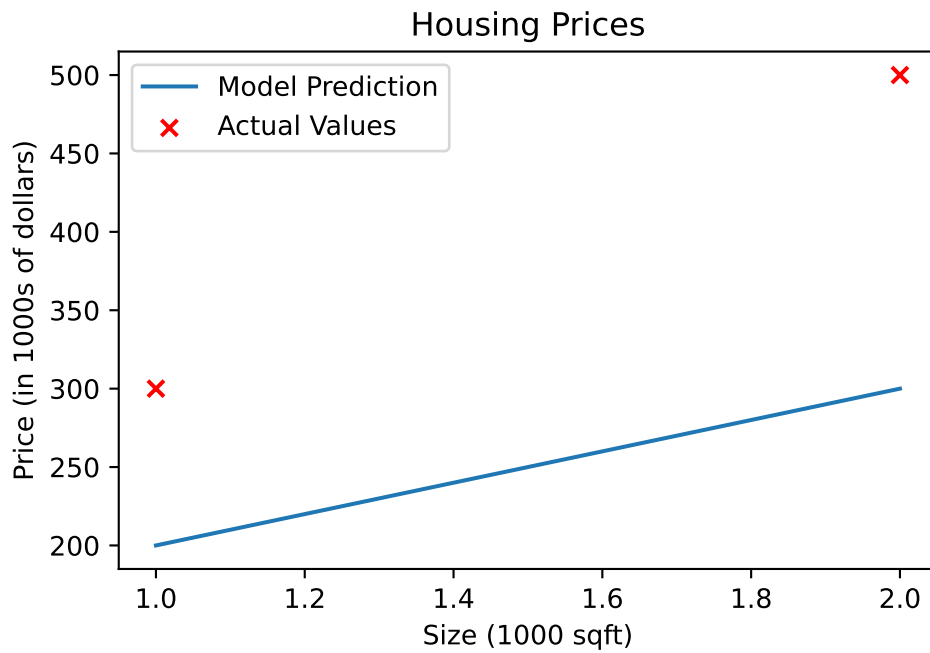
```

## 3 Control Run

```

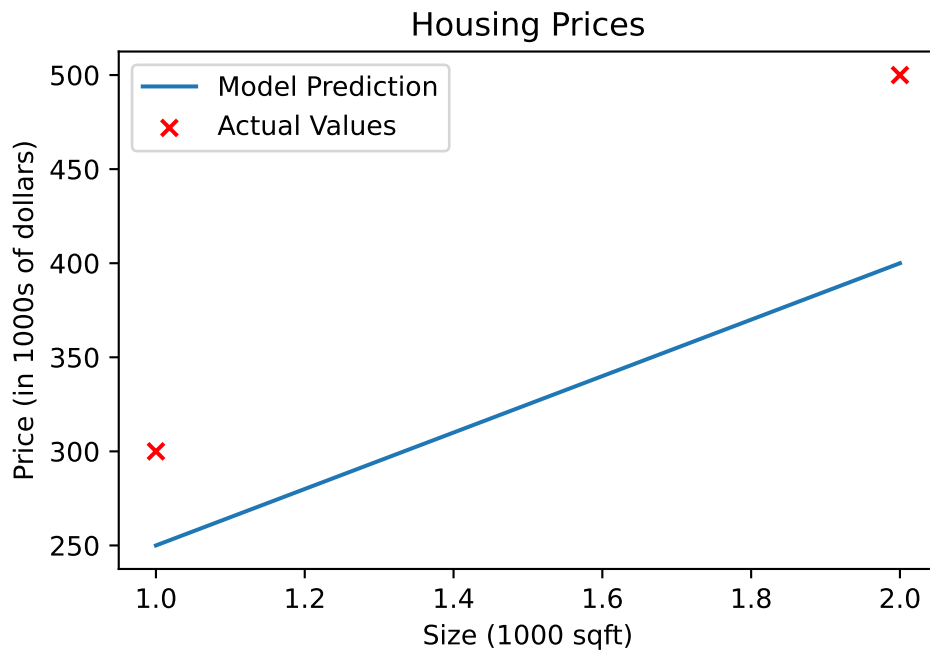
w = 100
b = 100
# Control
plot_model_prediction(x_train, y_train, w, b)

```



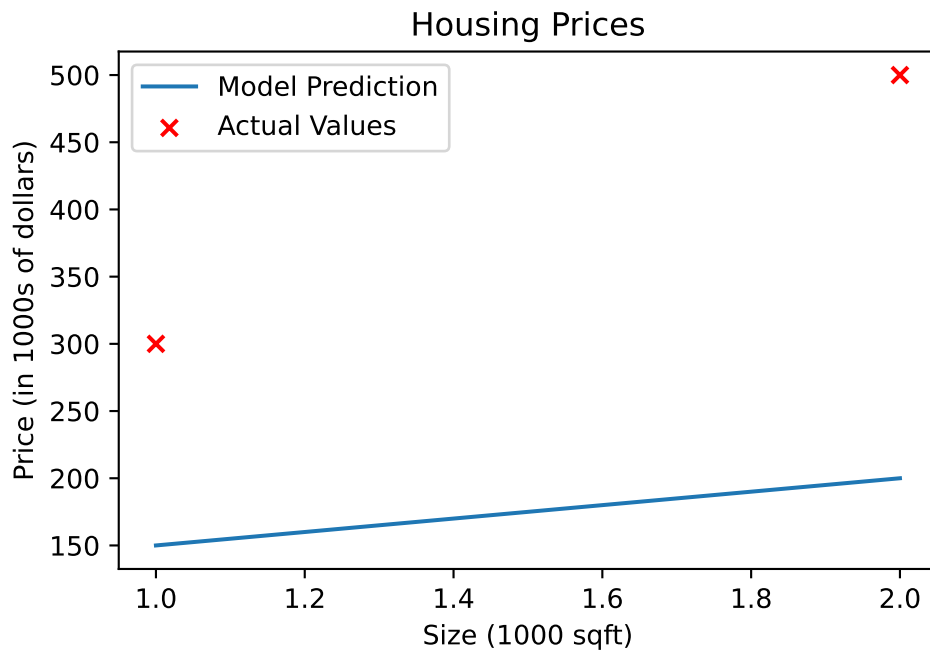
### 3.1 Trial 1 - Increase $w$ , maintain $b$

```
w = 150  
b = 100  
plot_model_prediction(x_train, y_train, w, b)
```



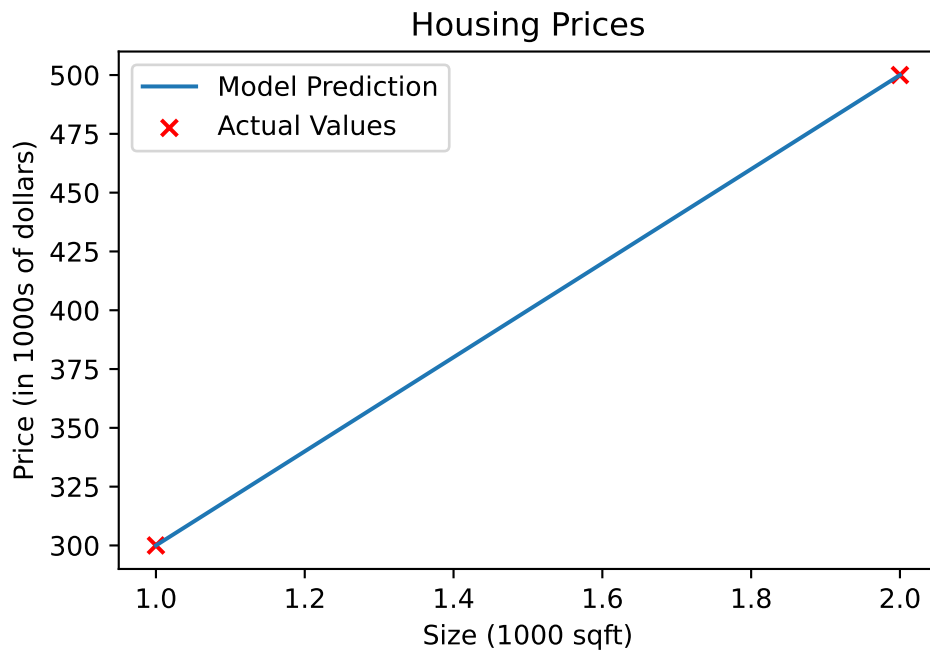
### 3.2 Trial 2 - Decrease $w$ , maintain $b$

```
w = 50  
b = 100  
plot_model_prediction(x_train, y_train, w, b)
```



### 3.3 Trial 3 - $w = 200$ , $b = 100$

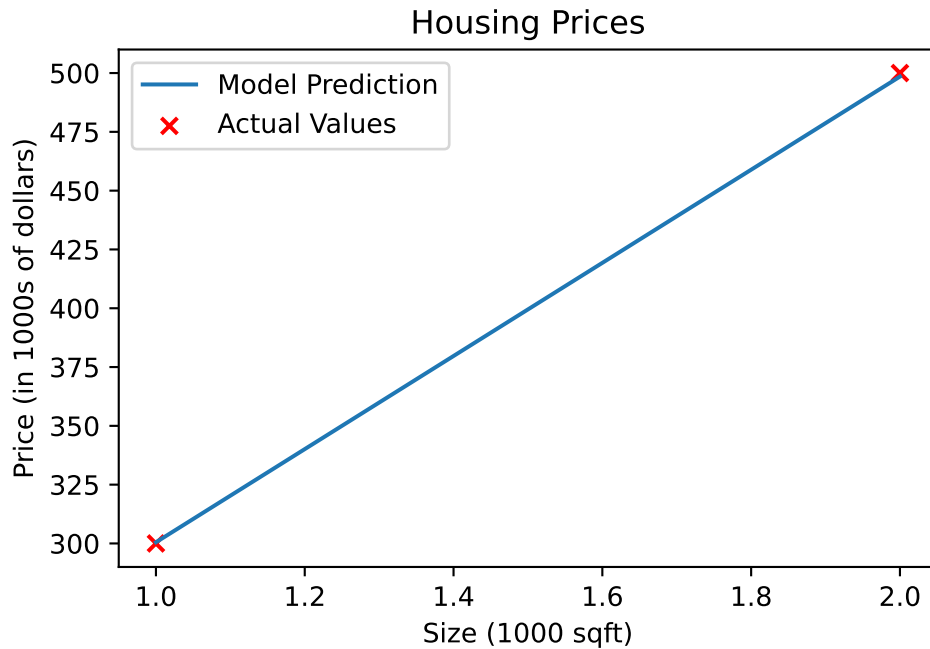
```
w = 200  
b = 100  
plot_model_prediction(x_train, y_train, w, b)
```



#### 3.4 Trial 4 - Looks Right, not optimum - Increase w by 98, increase b by 2.5

```
w = 198  
b = 102.5  
plot_model_prediction(x_train, y_train, w, b)
```





#### 4 Cost of a 1200 sqft house

```
w = 198
b = 102.5
x_i = 1.2
cost_1200sqft = w * x_i + b

print(f"Cost of a 1200 sqft house: ${cost_1200sqft}k")
```

Cost of a 1200 sqft house: \$340.1k

#### 5 Extra: Calculating Mean Square Error and Jwb Cost Function

```
def compute_cost(y: np.ndarray, f_wb: np.ndarray) -> float:
    """
    Computes the mean squared error
    Args:
```

```

    y (np.ndarray (m,)): Target values
    f_wb (np.ndarray (m,)): Model predictions
Returns:
    cost (float): Mean squared error
"""
m = y.shape[0]
cost = 0
for i in range(m):
    cost += (y[i] - f_wb[i]) ** 2
cost /= 2 * m
return cost

```

## 5.1 Compute Cost for Control Run

```

w = 100
b = 100
f_wb = compute_model_output(x_train, w, b)
cost = compute_cost(y_train, f_wb)
print(f"Cost for control run: {cost}")

```

Cost for control run: 12500.0

## 5.2 Compute Cost for Trial 1

```

w = 150
b = 100
f_wb = compute_model_output(x_train, w, b)
cost = compute_cost(y_train, f_wb)
print(f"Cost for trial 1: {cost}")

```

Cost for trial 1: 3125.0

## 5.3 Compute Cost for Trial 2

```
w = 50
b = 100
f_wb = compute_model_output(x_train, w, b)
cost = compute_cost(y_train, f_wb)
print(f"Cost for trial 2: {cost}")
```

Cost for trial 2: 28125.0

## 5.4 Compute Cost for Trial 3

```
w = 200
b = 100
f_wb = compute_model_output(x_train, w, b)
cost = compute_cost(y_train, f_wb)
print(f"Cost for trial 3: {cost}")
```

Cost for trial 3: 0.0

## 5.5 Compute Cost for Trial 4

```
w = 198
b = 102.5
f_wb = compute_model_output(x_train, w, b)
cost = compute_cost(y_train, f_wb)
print(f"Cost for trial 4: {cost}")
```

Cost for trial 4: 0.625