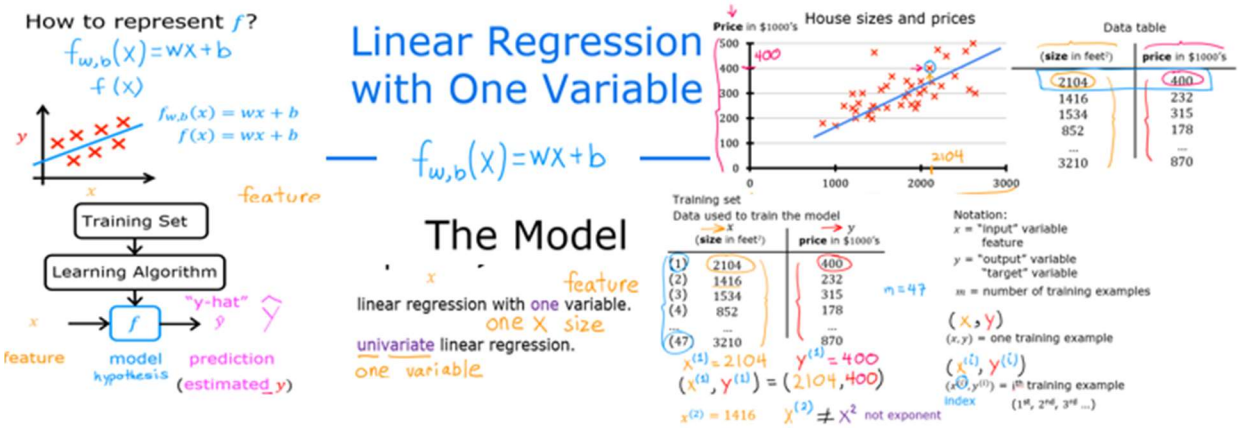
Task 1: Model Representation



Problem Statement

As in the lecture, you will use the motivating example of housing price prediction. This lab will use a simple data set with only two data points - a house with 1000 square feet(sqft) sold for $300,000 and a house with 2000 square feet sold for $500,000. These two points will constitute our *data* or *training set*. In this lab, the units of size are 1000 sqft and the units of price are 1000s of dollars.

Size (1000 sqft)	Price (1000s of dollars)
1.0	300
2.0	500

You would like to fit a linear regression model (shown above as the blue straight line) through these two points, so you can then predict price for other houses - say, a house with 1200 sqft.

Please run the following code to create your `x_train` and `y_train` variables. The data is stored in one-dimensional NumPy arrays.

```
# 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}")
```

Note: The course will frequently utilize the python 'f-string' output formatting described at the following link (https://docs.python.org/3/tutorial/inputoutput.html) when printing. The content between the curly braces is evaluated when producing the output.

Number of Training Examples 'm'

You will use `m` to denote the number of training examples. Numpy arrays have a `.shape` parameter. `x_train.shape` returns a python tuple with an entry for each dimension. `x_train.shape[0]` is the length of the array and number of examples as shown below.

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

```
One can also use the Python `len()` function as shown below.
```

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

Training example `x_i, y_i`

You will use (x_i, y_i) to denote the i-th training example. To access a value in a Numpy array, one indexes the array with the desired offset. For example the syntax to access location zero of `x_train` is `x_train[0]`.

Run the next code block below to get the i-th training example.

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

Plotting the Data

You can plot these two points using the `scatter()` function in the `matplotlib` library, as shown in the cell below.

- The function arguments `marker` and `c` show the points as red crosses (the default is blue dots).

You can use other functions in the `matplotlib` library to set the title and labels to display

```
# 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()
```

Model Function

As described in lecture, the model function for linear regression (which is a function that maps from `x` to `y`) is represented as

$$f_{w,b}\left(x^{(i)}\right) = wx^{(i)} + b$$

The formula above is how you can represent straight lines - different values of w and b give you different straight lines on the plot.

Let's try to get a better intuition for this through the code blocks below. Let's start with w = 100 and b = 100.

```
w = 100
b = 100
print(f"w: {w}")
print(f"b: {b}")
```

Now, let's compute the value of $f_{w,b}\left(x^{(i)}\right)$ for your two data points.

You can explicitly write this out for each data point as -

for $x^{(0)}$, `f_wb = w * x[0] + b`

for $x^{(1)}$, `f_wb = w * x[1] + b`

For a large number of data points, this can get unwieldy and repetitive. So instead, you can calculate the function output in a `for` loop. Kindly complete the `compute_model_output` function below.

Note: The argument description `(ndarray (m,))` describes a Numpy n-dimensional array of shape (m,). `(scalar)` describes an argument without dimensions, just a magnitude.

Note: `np.zero(n)` will return a one-dimensional numpy array with n entries.

```
def compute_model_output():
    """
    Computes the prediction of a linear model
    Args:
      x (ndarray (m,)): Data, m examples
```

```
      w,b (scalar)    : model parameters
    Returns
      y (ndarray (m,)): target values
    """
```

Now let's call the `compute_model_output` function and plot the output.

```
tmp_f_wb = compute_model_output(x_train, w, b,)

# Plot our model prediction
plt.plot(x_train, tmp_f_wb, c='b',label='Our Prediction')

# Plot the data points
plt.scatter(x_train, y_train, marker='x', c='r',label='Actual Values')

# 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.legend()
plt.show()
```

As you can see, setting w = 100$ and b = 100 does not result in a line that fits our data.

Challenge Question

Try experimenting with different values of w and b. What should the values be for a line that fits our data?

Prediction

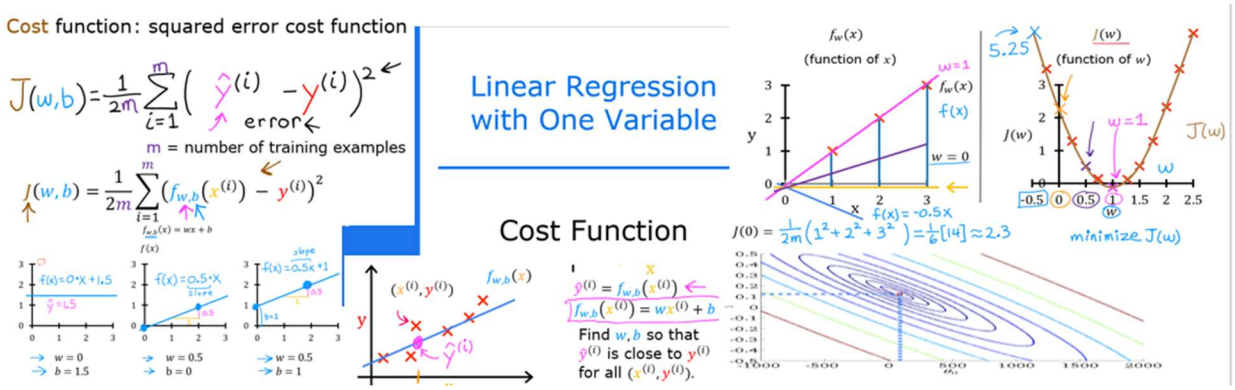
Now that we have a model, we can use it to make our original prediction. Let's predict the price of a house with 1200 sqft. Since the units of x are in 1000's of sqft, x is 1.2.

```
w = 200

b = 100

x_i = 1.2

cost_1200sqft = w * x_i + b


print(f"${cost_1200sqft:.0f} thousand dollars")
```

In this task you have learned:

1. Linear regression builds a model which establishes a relationship between features and targets
2. In the example above, the feature was house size and the target was house price
3. for simple linear regression, the model has two parameters w and b whose values are 'fit' using training data.
4. once a model's parameters have been determined, the model can be used to make predictions on novel data.

Task 2: Cost Function



Goals

In this task you will:

- implement and explore the `cost` function for linear regression with one variable.

Tools

In this lab we will make use of:

- NumPy, a popular library for scientific computing

- Matplotlib, a popular library for plotting data

- local plotting routines in the lab_utils_uni.py file in the local directory

```
import numpy as np

%matplotlib widget

import matplotlib.pyplot as plt

from lab_utils_uni import plt_intuition, plt_stationary,
plt_update_onclick, soup_bowl

plt.style.use('deeplearning.mplstyle')
```

Problem Statement

You would like a model which can predict housing prices given the size of the house.

Let's use the same two data points as before the previous lab- a house with 1000 square feet sold for $300,000 and a house with 2000 square feet sold for $500,000.

Size (1000 sqft)	Price (1000s of dollars)
1.0	300
2.0	500

```
x_train = np.array([1.0, 2.0])           #(size in 1000 square feet)
y_train = np.array([300.0, 500.0])           #(price in 1000s of dollars)
```

Computing Cost

The term 'cost' in this assignment might be a little confusing since the data is housing cost. Here, cost is a measure how well our model is predicting the target price of the house. The term 'price' is used for housing data.

The equation for cost with one variable is:

$$J(w,b) = \frac{1}{2m} \sum_{i=0}^{m-1} (f_{w,b}(x^{(i)}) - y^{(i)})^2$$

where

$$f_{w,b}(x^{(i)}) = wx^{(i)} + b$$

- $f_{w,b}(x^{(i)})$ is our prediction for example *i* using parameters *w,b*.

- $(f_{w,b}(x^{(i)}) - y^{(i)})^2$ is the squared difference between the target value and the prediction.

- These differences are summed over all the m examples and divided by `2m` to produce the cost, *J(w,b)*.

Note, in lecture summation ranges are typically from 1 to m, while code will be from 0 to m-1.

Task 2 Challenge Question

Complete the following code that calculates cost by looping over each example. In each loop:

- `f_wb`, a prediction is calculated

- the difference between the target and the prediction is calculated and squared.

- this is added to the total cost.

```
def compute_cost( ):
    """
    Computes the cost function for linear regression.

    Args:
      x (ndarray (m,)): Data, m examples
      y (ndarray (m,)): target values
      w,b (scalar)    : model parameters

    Returns
        total_cost (float): The cost of using w,b as the parameters
for linear regression
               to fit the data points in x and y
    """
    # number of training examples
```

Cost Function Visualization- 3D

You can see how cost varies with respect to both `w` and `b` by plotting in 3D or using a contour plot. It is worth noting that some of the plotting in this course can become quite involved. The plotting routines are provided in lab_utils_uni.py in the lab folder

Larger Data Set

It's use instructive to view a scenario with a few more data points. This data set includes data points that do not fall on the same line. What does that mean for the cost equation? Can we find w, and b that will give us a cost of 0?.

```
x_train = np.array([1.0, 1.7, 2.0, 2.5, 3.0, 3.2])
y_train = np.array([250, 300, 480,  430,   630, 730,])
```

In the contour plot, click on a point to select `w` and `b` to achieve the lowest cost. Use the contours to guide your selections. Note, it can take a few seconds to update the graph.

```
plt.close('all')
fig, ax, dyn_items = plt_stationary(x_train, y_train)
updater = plt_update_onclick(fig, ax, x_train, y_train, dyn_items)
```

Above, note the dashed lines in the left plot. These represent the portion of the cost contributed by each example in your training set. In this case, values of approximately w=209 and b=2.4 provide low cost. Note that, because our training examples are not on a line, the minimum cost is not zero.

Convex Cost surface

The fact that the cost function squares the loss ensures that the 'error surface' is convex like a soup bowl. It will always have a minimum that can be reached by following the gradient in all dimensions. In the previous plot, because the w and b dimensions scale differently, this is not easy to recognize. The following plot, where w and b are symmetric, was shown in lecture:

```
soup_bowl()
```

In this task you have learned the following:

1. The cost equation provides a measure of how well your predictions match your training data.
2. Minimizing the cost can provide optimal values of w, b.