Artificial Intelligence II

Lesson 5 - Regression

First Code Academy

Today’s Plan

Teach Back	00 - 5 min
Regression	10 - 15 min
Gradient Descent	15 - 20 min
Quiz	20-25 min
Break	25 - 28 min
Project - Curve Fitting	28 - 55 min
Lesson Recap	55 - 60 min

Teach Back

______ algorithms always make the **locally** best choice, i.e. the best one at that moment

When we store the solutions, we avoid solving ______ multiple times.

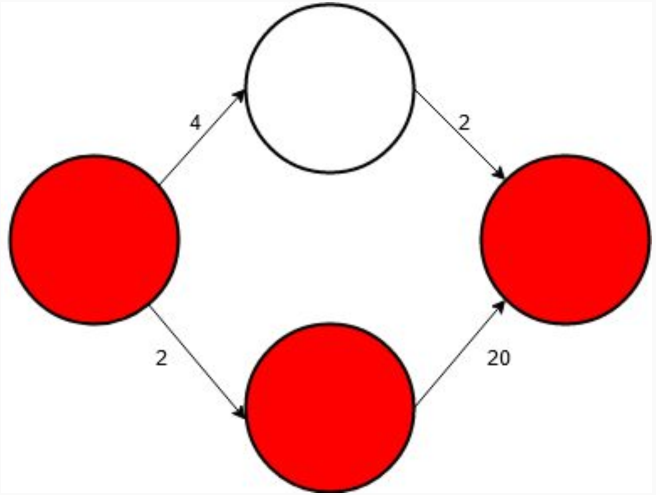
What did we learn last time?

Greedy Algorithms

Greedy algorithms choose the best option **locally**, which might not be the best option **globally**

Greedy Algorithms

Do not always lead to an optimal solution



Dynamic Programming

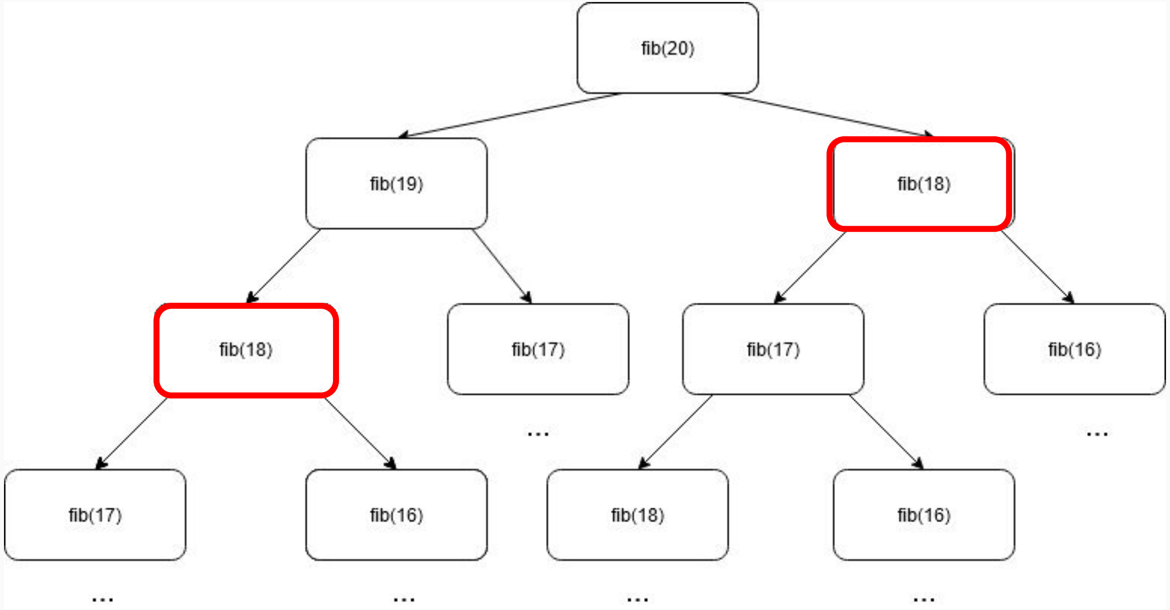
Relies on solving smaller versions of the problem to solve the larger version

```
fib(20) = fib(19) + fib(18)
```

Smaller versions of the problem

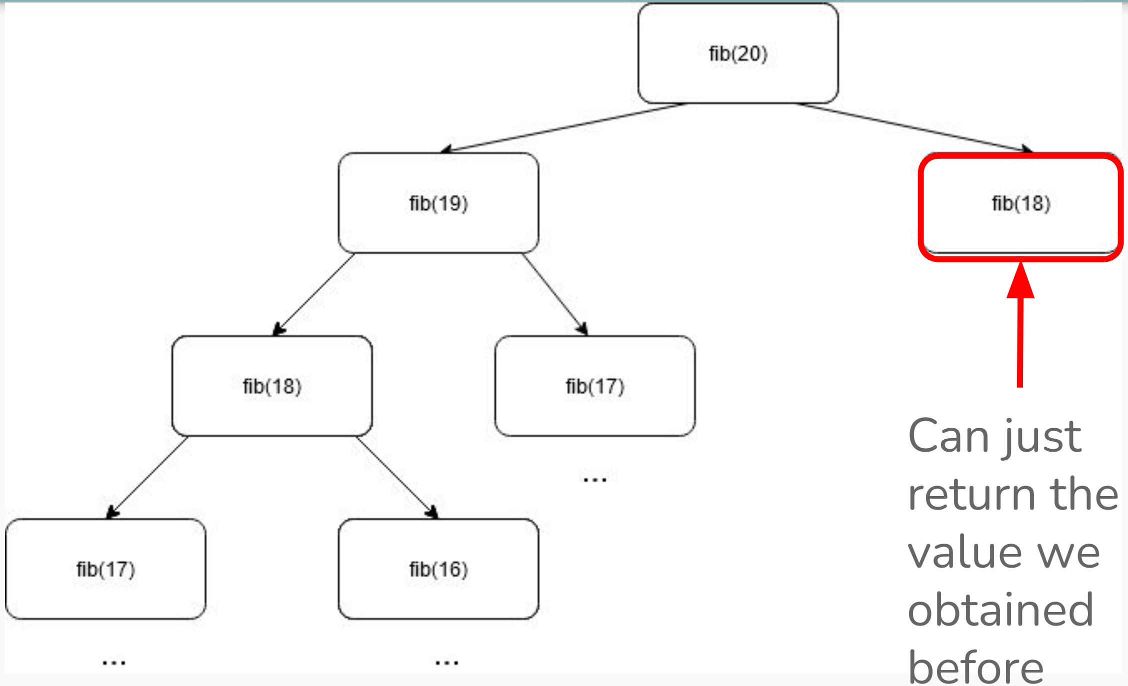
Dynamic Programming

Not efficient to calculate same result over and over



Dynamic Programming

If we store our solution at every step, we can cut down on a lot of extra work.



Can just return the value we obtained before

Key Terms

Regression

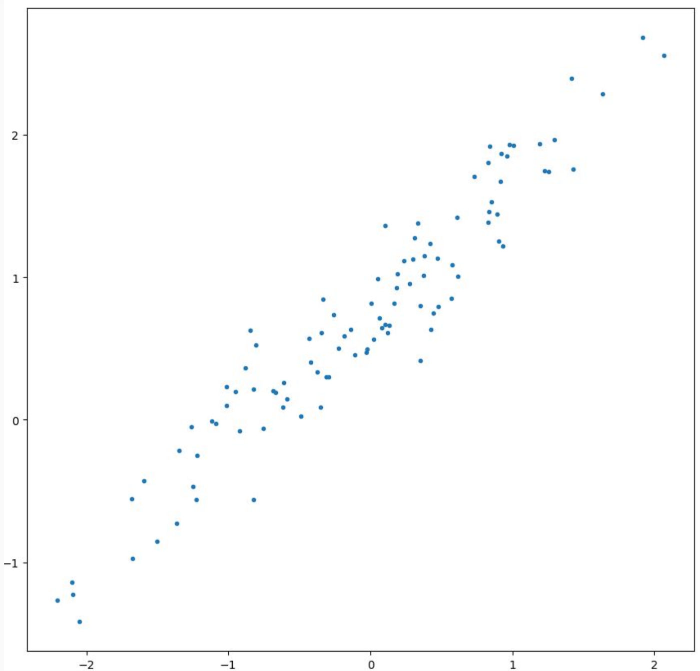
Gradient Descent

Regression

Regression

Regression deals with finding the relation between two variables

We can find the line that best describes a set of data points



Lines

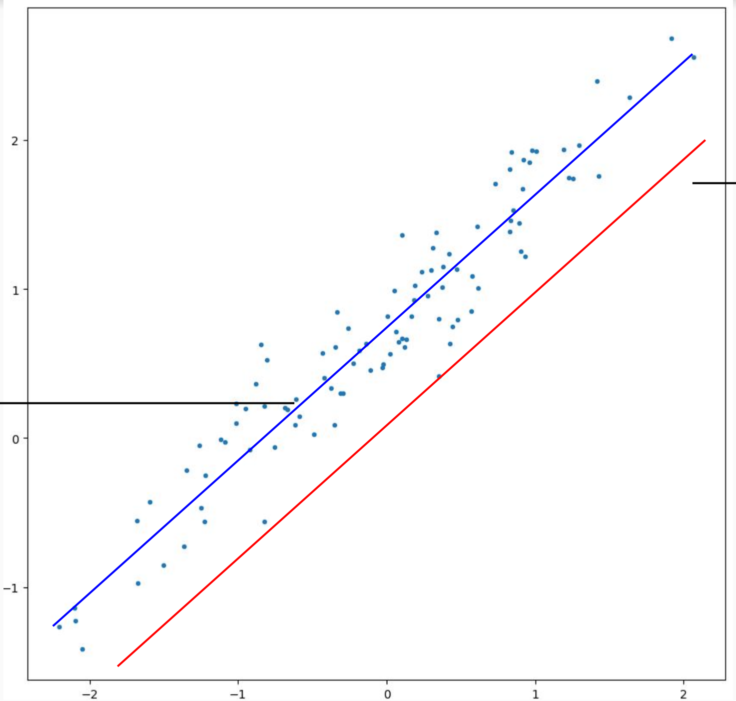
The equation of a line is

$$\underbrace{y}_{\text{height}} = \underbrace{m}_{\text{slope}}\overbrace{x}^{\text{width}} + \underbrace{b}_{\text{intercept}}$$

Regression

What is the best line?

But this line goes through more points



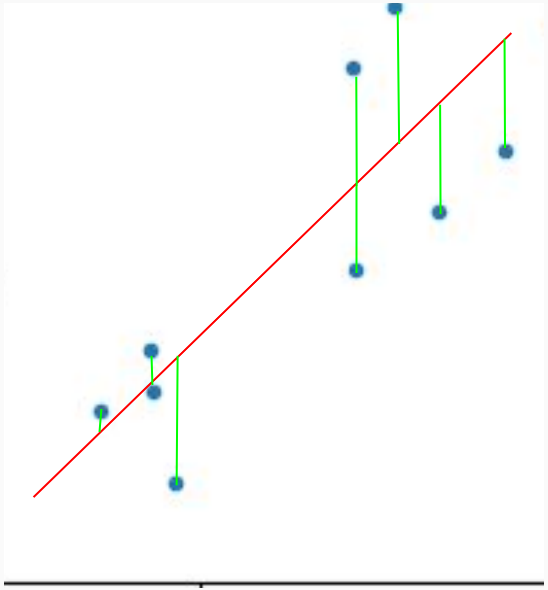
This line kind of follows the same pattern as the points

Regression

We can measure the error, or the difference between the line and the point.

We find the line that has the smallest error!

This is the **loss** function



Regression

We need the optimal slope and intercept of the line with the smallest error

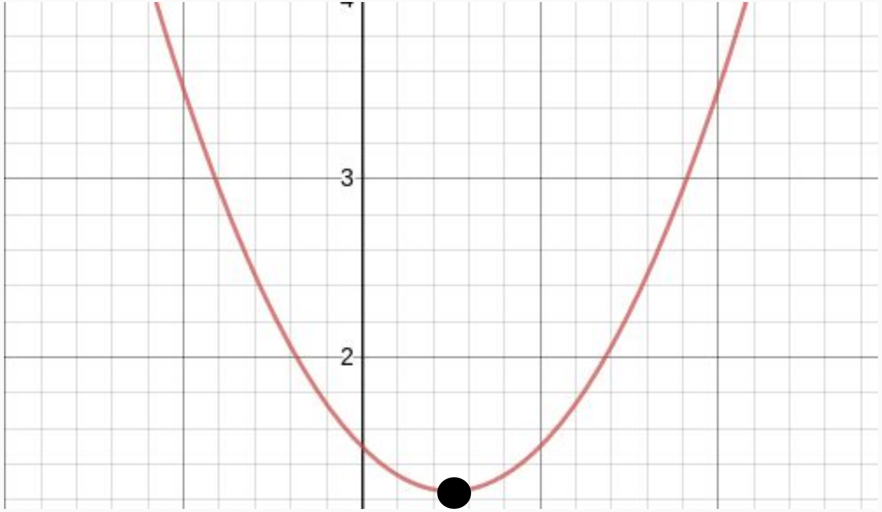
How can we find it?

Gradient Descent

Gradient Descent

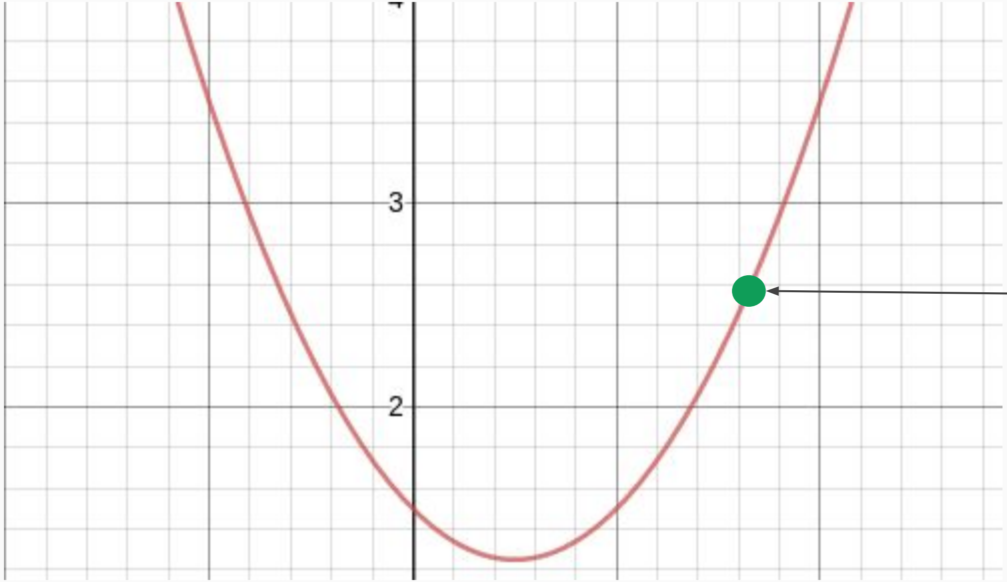
Using gradient descent, we try and adjust our slope and intercept to minimize the loss function

The loss function is the function that measures the error.



Gradient Descent

At first, we guess a random slope and intercepts!

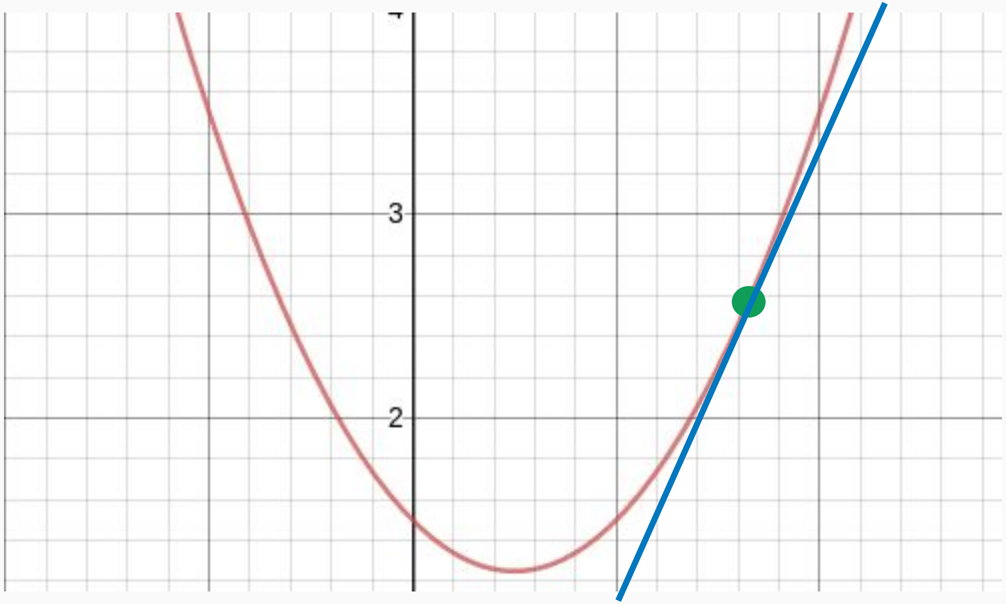


Our error is probably very high at first!

Gradient Descent

We move in the direction that's "going down"

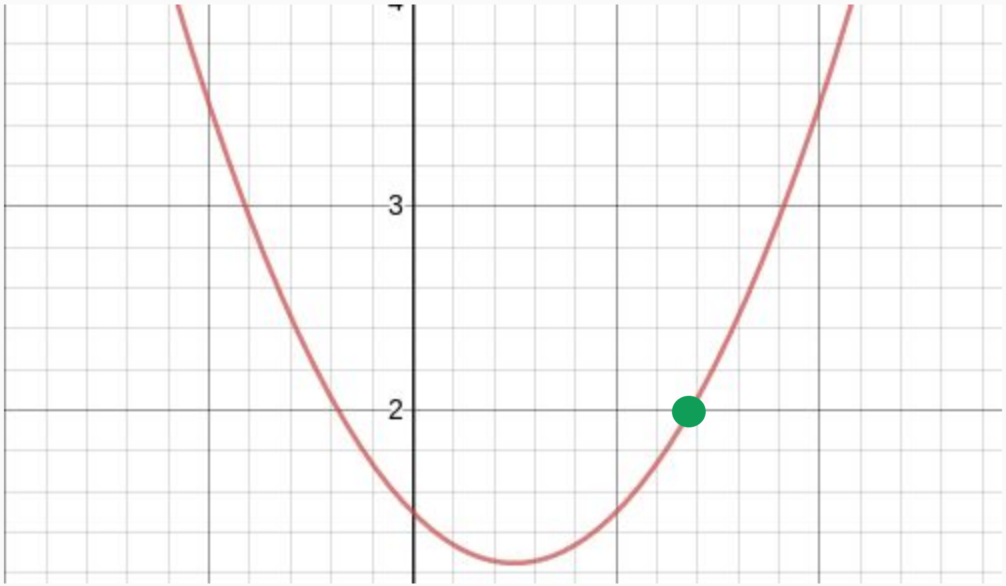
The **derivative** of the loss tells us the change of slope at a point!



Gradient Descent

We take a small step in the "downhill" direction.

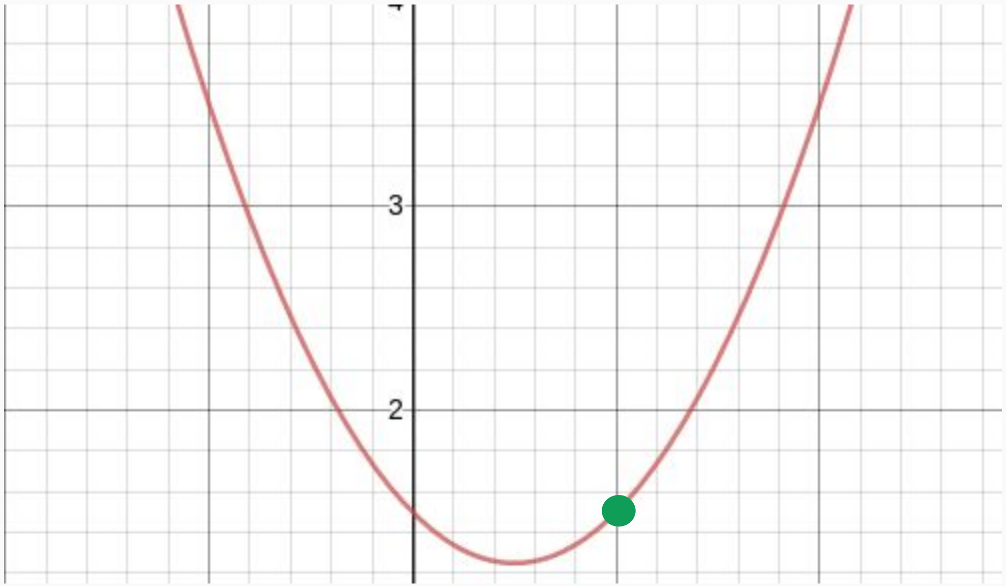
Imagine a ball rolling down a hill!



Gradient Descent

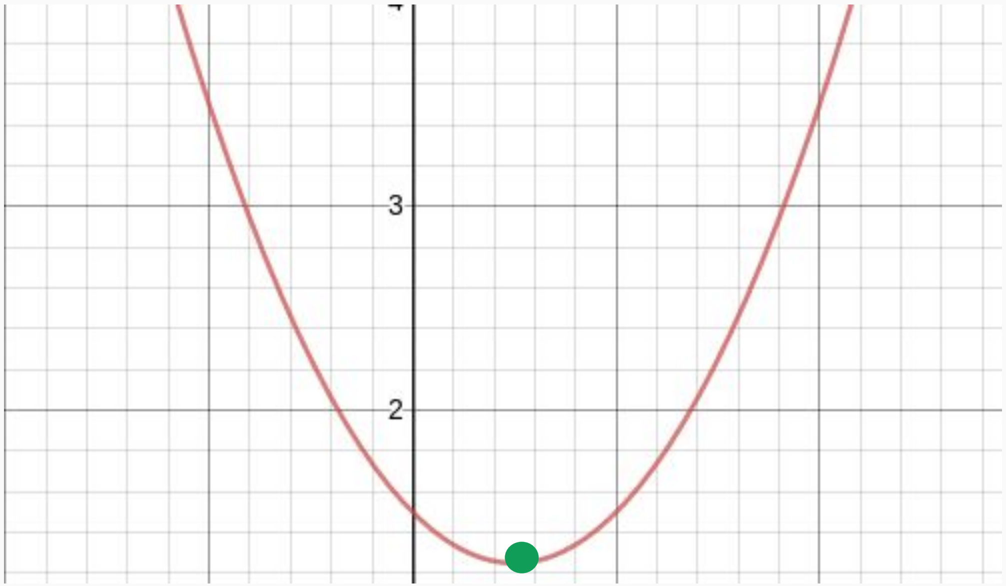
Maybe we have to adjust our guess a couple times.

Notice we get closer to the bottom every time



Gradient Descent

Eventually, our guess is not far off from the minimum!



Gradient Descent

At every step, we update our guess:

New guess = old guess - (decreasing slope * learning rate)

Recap of Gradient Descent

1. Pick random guesses for our slope and intercept
2. Calculate the direction in which the cost decreases using the **derivative** (rate of change in slope)
3. Update our slope and intercept in this direction
4. Repeat steps 2 and 3 until we reach the minimum.

Quiz: bit.ly/FCA_AI_Quiz

Project: Line Fitting

Get the starter file

http://bit.ly/FCA_AI2_Starter

Download Libraries

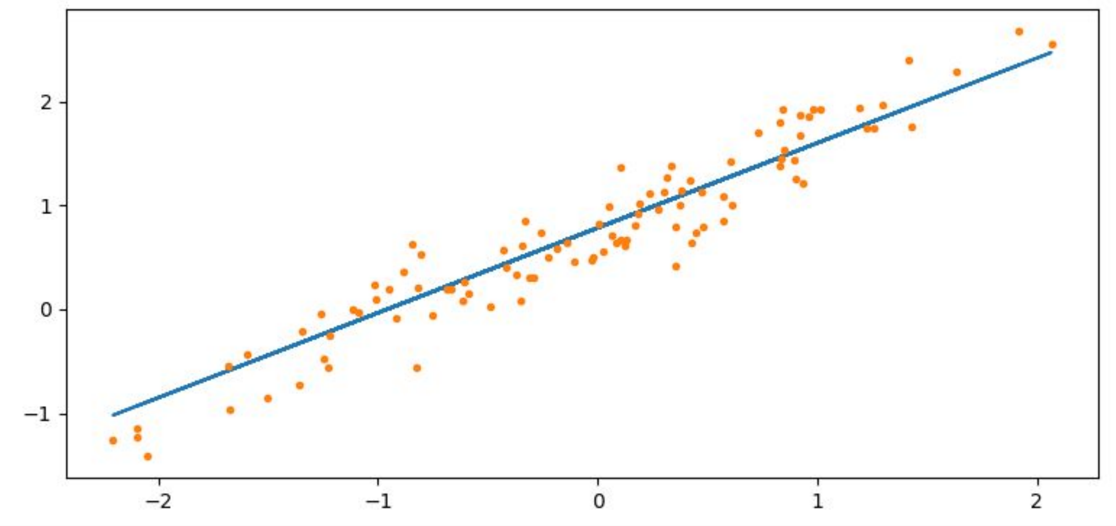
For this project we will need the `pandas`, `matplotlib`, and `numpy` libraries.

Install them by typing in your terminal:

```
pip install numpy matplotlib pandas
```

Objective

Given a training dataset, find the line that best describes that dataset and apply it to a testing dataset!



Steps

1. Load the training data set
2. Find the best slope and intercept for the training set
3. See how well our model works on data it hasn't seen before (testing set)

Load the Training Data

Load a bunch of (x, y) pairs to train our model

```
    # Training Data
    train_df = pd.read_csv("train_data.csv")
    x_train = train_df['x_train'].to_numpy().reshape(-1, 1)
    y_train = train_df['y_train'].to_numpy().reshape(-1, 1)

    # Create a model class
```

Fill in Class Methods

In `LinearRegression()` class, we fill in the `__init()__` method

```python
class LinearRegression():
    def __init__(self, learning_rate=0.001, batch_size=32, epochs=100):
        self.learning_rate =  learning_rate # Size of the step we take
        self.batch_size = batch_size        # See the data points in batch
        self.epochs = epochs                # Times we adjust our predicti

```

Calculate the Optimal Parameters!

First, we set our slope and intercept to random numbers

```
    # Calculate the best line
    def fit(self, X, Y):
        # First we set our slope and intercept to random value
        self.slope = np.random.rand(X.shape[1])
        self.intercept = np.random.rand(1)
        num_points = X.shape[0]

```

Calculating the Optimal Parameters!

We make a prediction and then adjust our direction with the gradient

```
for i in range(0, num_points, self.batch_size):

    # Get the coordinates for the current batch
    x = X[i: i + self.batch_size]
    y = Y[i: i + self.batch_size]

    # Get the prediction and loss
    y_pred = (x * self.slope) + self.intercept
    loss = y_pred - y

    # Get the gradient of slope and intercept
    slope_gradient = 2 * np.sum(x * loss) / len(x)
    int_gradient = 2 * np.sum(loss)
```

Update Our Slope and Intercept

We move our slope and intercept in the direction that the error will decrease

```
int_gradient = 2 * np.sum(loss)

# Update our guess for slope and intercept
self.slope = self.slope - self.learning_rate * slope_gradient
self.intercept = self.intercept - self.learning_rate * int_gradient
```

Create a Model and Fit Dataset

In `main()` we create our model and find best fit for the training dataset.

```
    # Create a model class
    model = LinearRegression()

    # Calculate the best slope and intercept for training set
    model.fit(x_train, y_train)

```

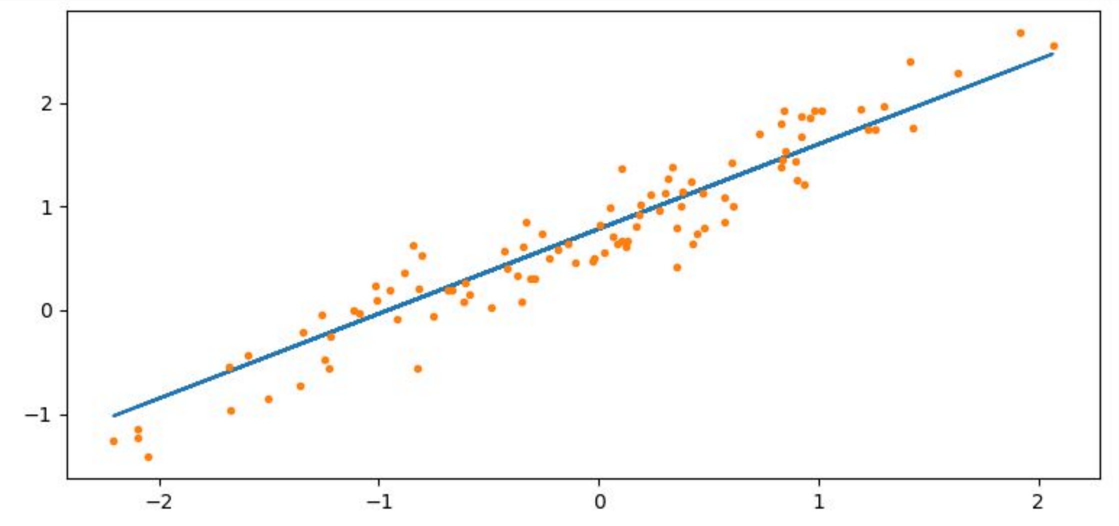
Testing Set

Predict the line for the testing set and plot it.

```python
    # Predict the line going through the testing data
    y_pred = model.predict(x_test)
    plt.plot(x_test, y_test, '.')
    plt.title("Testing data")
    plt.plot(x_test, y_pred, '-')
    plt.show()
```

Try it out!

You should see this same image!



That’s it for today!

Artificial Intelligence II

Lesson 5 - Regression

First Code Academy