**Module 15**

**Gradient Descent and Optimization**

1. Compute x and/or y for each iteration of gradient descent
2. Explain how learning rate and starting guess affect the convergence of gradient descent
3. Optimize a single-parameter linear regression model from scratch
4. Recognize convex one-dimensional and two-dimensional functions
5. Compute the gradient of a two-dimensional function
6. Use gradient descent to optimize a nonlinear two-dimensional regression model
7. Use stochastic gradient descent to optimize a nonlinear two-dimensional regression model
8. Compare the convergence behavior of gradient descent with stochastic gradient descent
9. Identify the degrees of bias and variance in a model
10. Identify the relationship between bias, variance, and model complexity

* [Video Transcripts](https://student.emeritus.org/courses/4765/files/3390438?wrap=1)
* [Download Video Transcripts](https://student.emeritus.org/courses/4765/files/3390438/download?download_frd=1)
* [Quick Reference Guide](https://student.emeritus.org/courses/4765/files/3390449?wrap=1)

**Program learning outcomes addressed this module:**

* Apply real-world tools to model and analyze real-world data
* Communicate foundational concepts about AI/ML
* Draw useful conclusions from real-world data
* Identify the best ML model to solve a problem (Models: Classification, regression, time series analysis)

**Optimizing Linear Regression**

As previously discussed, linear regression assumes only one independent variable and a linear relationship between the independent (X) and dependent (Y) variables.Linear regression determines the relationship between the two variables by fitting a linear equation to the observed data. The best fit can be defined by the hypothesis equation, where w0 and w1 are weighted so that they are optimized to fit the best line and reduce cost (loss). Two primary methods are used to minimize the cost: Gradient descent and normal equation. Gradient descent can find the value of w0 and w1 using an iterative process to minimize the overall cost. In contrast, normal equation can find the weights (w) for which the cost is minimal.

**The Bias–Variance Tradeoff**

Prediction errors can be broken down into two main subcomponents: bias and variance. Bias and variance are key parameters that need to be tuned while training a machine learning model.

**Bias Error**

A bias error is the difference between a model’s predictions and the actual value. In this type of error, the model ignores training data and oversimplifies the model without learning the patterns.

**Variance Error**

A model’s ability to predict a given data point or value tells you how widely your data is spread. During this type of error, the model pays a lot of attention to training data, to the point that it memorizes the data rather than learning from it. As a result, models with a high variance error have difficulty generalizing on unseen data.

The bias–variance tradeoff describes the tension between bias-introduced and variance-introduced errors.

**Glossary**

**Bias Error**

The difference between a model’s predictions and the actual values

**Convexity**

A measure of the curvature, or the degree of the curve, where a line drawn between two points will have all points between the two endpoints under the line

**Gradient Descent**

An algorithm used to find a local minimum/maximum of a given function

**Stochastic Gradient Descent**

A technique that is an approximation of gradient descent

**Variance Error**

A model’s ability to predict a given data point or value that tells you how widely the data is spread

Gradient Descent is used in Liner Regression models, not for any other methodology!

**Notes:**

**Batch gradient descent**, also known as vanilla gradient descent, calculates the errors for each example in the training dataset, but the model is only updated once all training examples have been evaluated. The whole process is referred to as a training epoch.

**Stochastic gradient descent (SGD**) updates the parameters for every training example in the dataset, meaning each example’s parameters are updated one at a time. In some cases, this can make SGD faster than batch gradient descent.

**Mini-batch gradient descent** is the preferred method because it combines the concepts of batch gradient descent and SGD. It splits the training dataset into small batches and updates each batch individually. A balance is thus created between the robustness of SGD and the efficiency of batch gradient descent.

convex aka concave up!

M = (y2 - y1) / (x2 - x1)

t1s = ex2.iloc[:, 1]

t0s = ex2.iloc[:, 0]

theta1 = np.linspace(0, 20, 100)

theta0 = np.linspace(0, 20, 100)

T0, T1 = np.meshgrid(theta0, theta1)

fig = plt.figure(figsize=(16, 6))

ax = fig.add\_subplot(1, 2, 1, projection = '3d')

ax.plot3D(t0s, t1s, mse(t0s, t1s), '-->', color = 'red')

ax.plot\_wireframe(T0, T1, mse(T0, T1), alpha = 0.2)

ax2 = fig.add\_subplot(1, 2, 2)

ax2.contour(T0, T1, mse(T0, T1), levels = 30)

ax2.plot(t0s, t1s, mse(t0s, t1s), color = 'red')

ax2.set\_xlim(0, 20)

ax2.set\_ylim(0, 20)

**Module Issues:**

**Codio Activity 15.7 Problem 3**: Hidden Test uses *mse\_grad* student solution!

***for*** *i* ***in*** *range(1000):*

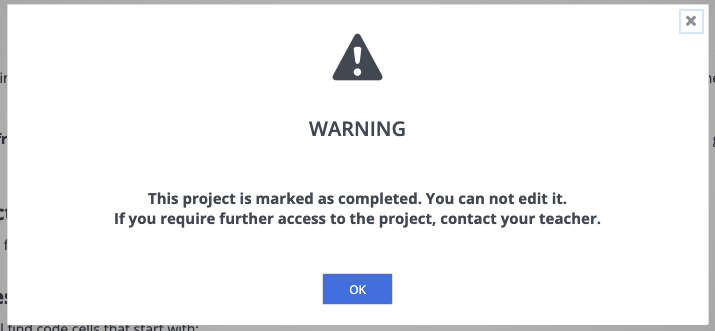
*thetas\_.append(theta\_)*

*theta\_ = theta\_ - lr\*mse\_grad(theta\_, X\_, y)*

**Codio Activity 15.8** Plot broken projection = “3”d

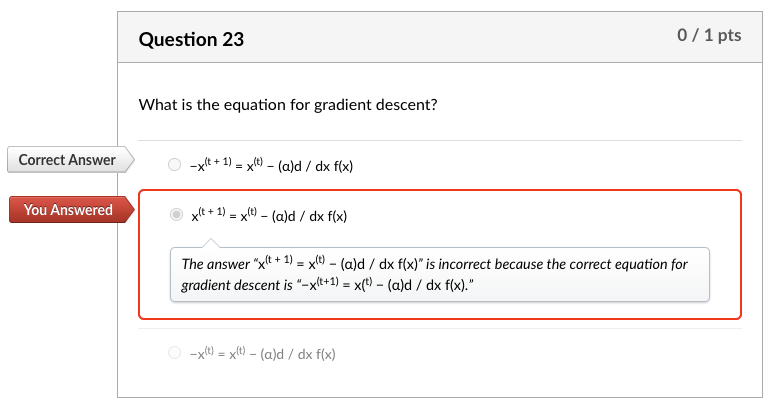
**Codio Activity 15.9 Problem 2**: typo in print(***sgd\_defaults***)

**Codio Activity 15.10** Reset option is not given

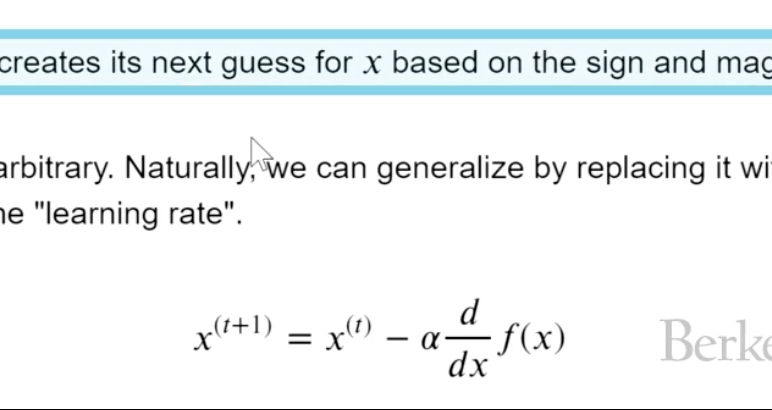


**End of Module Quiz Problems:**

The answer sheet template is wrong for a question, please see the attachment and correct the answer template as well as my grade. Thanks.



What is the equation for gradient descent?



**Quizes:**

Gradient descent is a technique used to minimize functions. : True

*You are correct! The answer “*True*” is correct because the gradient descent (GD) is an iterative first-order optimization algorithm used to find a local minimum/maximum of a given function.*

Consider the following Python function:

minimize(arbitrary,x0)

What is the second constructor used as? : The starting point for the function to minimize

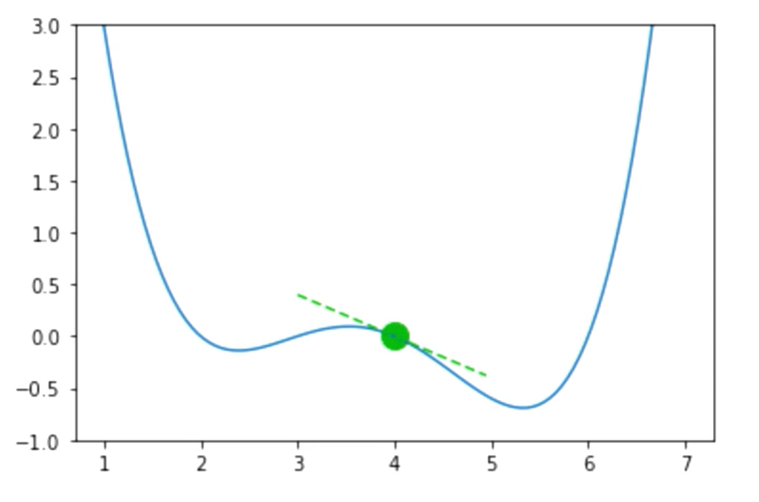
*You are correct! The answer “*The starting point for the function to minimize*” is correct because it represents the initial guess as the starting value to minimize the function.*

If the derivative of a function is negative, that means the function is (blank). : decreasing

*You are correct! The answer “*decreasing*” is correct because a function is decreasing if a derivative of that function is negative.*

In gradient descent, the point where there is a minima or maxima of the function is where the derivative is (blank). : zero

*You are correct! The answer “z*ero*” is correct because when the derivative of a function is zero that point is the minima or maxima of the function.*



Given the above graph, suppose the guess for a function input is x = 4, f(x) = 0.0, and f’(x) = −0.4. Should you increase or decrease x for your next guess? : Increase

*You are correct! The answer “*Increase*” is correct because if the derivative of a function is negative, that means the function is decreasing, so you increase the x.*

What is the equation for gradient descent? : x(t + 1) = x(t) − (α)d / dx f(x)

*You are correct! The answer “*x(t + 1) = x(t) − (α)d / dx f(x)*” is correct because this is the equation for gradient descent.*

The value alpha (α) in the gradient descent function is known as learning rate. : True

*You are correct! The answer “*True*” is correct because it is known as the learning rate.*

The learning rate alpha (α) captures how quickly gradient descent learns the minimum. A small alpha moves slowly but has a high chance of overshooting. : False

*You are correct! The answer “*False*” is correct because a small alpha means small jumps toward the minimum, which have a lower chance of overshooting.*

Gradient descent always gives the global minimum. : False

*You are correct! The answer “*False*” is correct because if the function has a local minimum and the algorithm’s initial guess (which is the starting point) is close to the minimum, it can get stuck at the local minimum.*

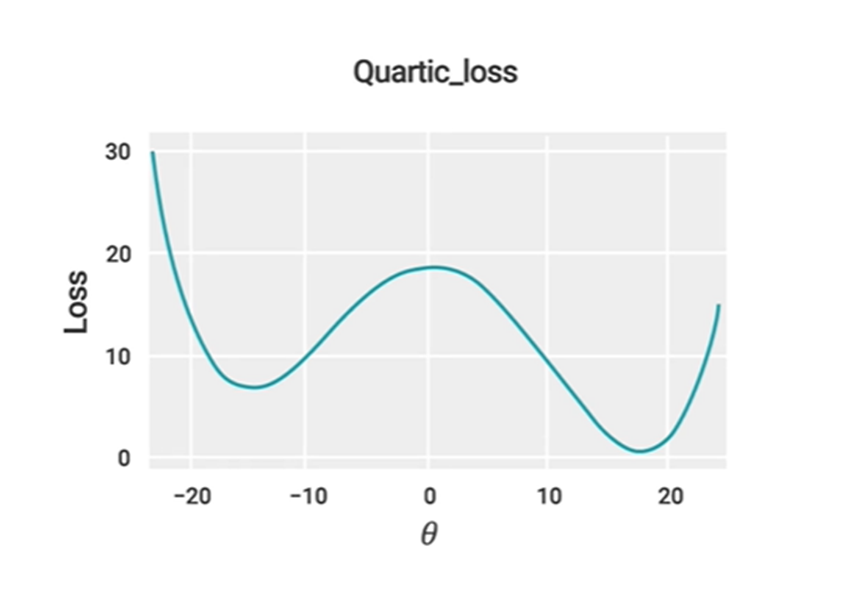
The gradient descent is guaranteed to find the global minimum if the function has convexity. : True

*You are correct! The answer “*True*” is correct because the gradient descent is guaranteed to find the global minimum if the function has convexity.*

Under what circumstances is the loss function f said to be convex? : If a line is drawn between two points on a curve, all values on the curve must be on or below the line

*You are correct! The answer “*If a line is drawn between two points on a curve, all values on the curve must be on or below the line*” is correct because this is the condition for a function to be convex.*

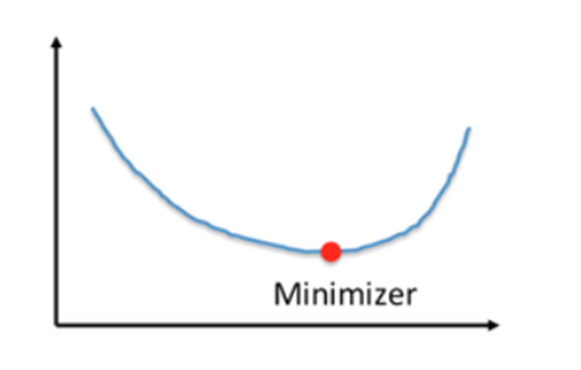
Consider the following the graph:



Is it convex or nonconvex? : Nonconvex

*You are correct! The answer “*Nonconvex*” is correct because the condition “if a line is drawn between two points on a curve, all values on the curve must be on or below the line” must be satisfied for a function to be convex. This plot does not satisfy this condition.*

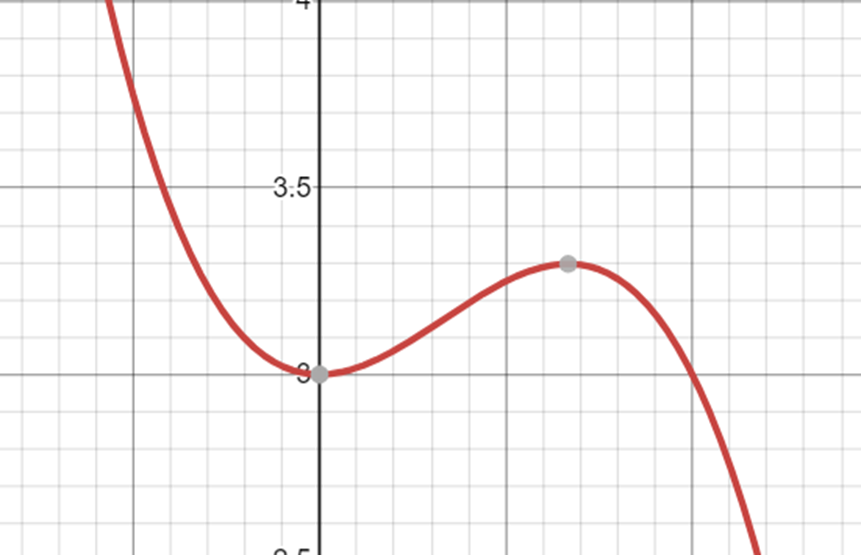
Consider the following graph:



Is it convex or nonconvex? : Convex

*You are correct! The answer “C*onvex*” is correct because the condition “if a line is drawn between two points on a curve, all values on the curve must be on or below the line” must be satisfied for a function to be convex. This plot satisfies this condition.*

Consider the following graph:



Is it convex or nonconvex?

*You are correct! The answer “*Nonconvex*” is correct because the condition “if a line is drawn between two points on a curve, all values on the curve must be on or below the line” must be satisfied for a function to be convex. This plot does not satisfy this condition.*

The function for mean squared error can be represented as mean((y\_hat - y\_obs)\*\*2). : True

*You are correct! The answer “*True*” is correct because this is the function for mean squared error.*

For a model Tip = θ1 + θ2 \* bill, the MSE loss function cannot be used to predict the optimal θ1 and θ2 values. : False

*You are correct! The answer “*False*” is correct because the MSE loss function can be used to predict the optimal θ1 and θ2 values.*

**def**mse\_loss(theta, X, y\_obs):     y\_hat **=** theta[0] **\*** X.iloc[:, 0] **+** theta[1] **\*** X.iloc[:, 1]

*#can also write in matrix form:* *#y\_hat = X @ theta*

**return**np.mean((y\_hat **-** y\_obs) **\*\*** 2)

From the function in the Python code block, determine the correct call statement for the function. : mse\_loss(np.array([1,2]),X,y\_obs)

*You are correct! The answer “*mse\_loss(np.array([1,2]),X,y\_obs)*” is correct because this is the correct function call statement.*

**def**mse\_loss(theta, X, y\_obs):     y\_hat **=** theta[0] **\*** X.iloc[:, 0] **+** theta[1] **\*** X.iloc[:, 1]

*#can also write in matrix form:* *#y\_hat = X @ theta*

**return**np.mean((y\_hat **-** y\_obs) **\*\*** 2)

From the function in the Python code block, determine the correct representation for a single argument function. : def mse\_loss\_single\_arg(theta):

    X = tips\_with\_bias[["bias", "total\_bill"]]

    y\_obs = tips["tip"]

    return mse\_loss(theta, X, y\_obs)

*You are correct! The answer “*def mse\_loss\_single\_arg(theta):

    X = tips\_with\_bias[["bias", "total\_bill"]]

    y\_obs = tips["tip"]

    return mse\_loss(theta, X, y\_obs)*” is correct because this is the correct representation for a single argument function.*

Given a function f(θ0,θ1) with two variables θ0 and θ1, what would the formula for the gradient of this 2D function be? : df / d(θ0)i + df / d(θ1)j

*You are correct! The answer “*df / d(θ0)i + df / d(θ1)j*” is correct because the gradient of the 2D function is a partial derivative with respect to variable one and a partial derivative with respect to variable two.*

To reduce the loss function, how should the θ values be adjusted? *Check all that apply.* : *“*Increase all values of θ that have a negative partial derivative*” and “*Decrease all values of θ that have a positive partial derivative.*”*

*You are correct! The answers “*Increase all values of θ that have a negative partial derivative*” and “*Decrease all values of θ that have a positive partial derivative*” are correct because these are the techniques to adjust the value of*θ *to reduce loss function.*

Consider the following two-dimensional linear regression model:

fθ→(x→)=(x→)Tθ→=θ0x0+θ1x1

What would the squared loss be for the single prediction of a linear regression model provided below? ℓ(θ→,x→,yi) : (yi − θ0x0 − θ1x1)2

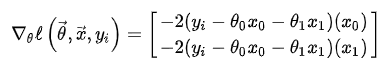
*You are correct! The answer “(yi − θ0x0 − θ1x1)2” is correct because this is the correct representation of squared loss for a single prediction of a linear regression model.*(yi − θ0x0 − θ1x1)2

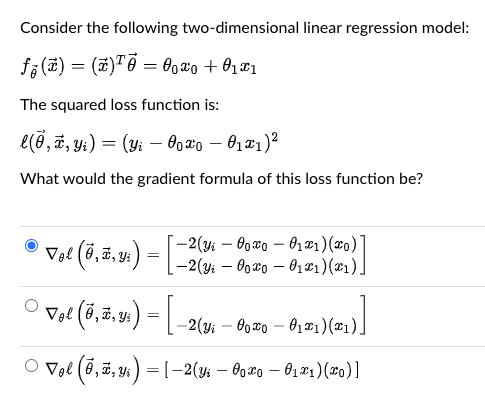
Consider the following two-dimensional linear regression model:

fθ→(x→)=(x→)Tθ→=θ0x0+θ1x1

The squared loss function is:ℓ(θ→,x→,yi)=(yi−θ0x0−θ1x1)2

What would the gradient formula of this loss function be? :





*You are correct! The answer “*

∇θℓ(θ→,x→,yi)=[

|  |
| --- |
| **−2(y**i**−θ**0**x**0**−θ**1**x**1**)(x**0**)** |
| **−2(y**i**−θ**0**x**0**−θ**1**x**1**)(x**1**)** |

]

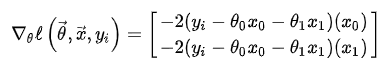
*” is correct because this is the correct formula for the gradient of this loss function.*

Consider the given gradient formula:

∇θℓ(θ→,x→,yi)=[

|  |
| --- |
| **−2(y**i**−θ**0**x**0**−θ**1**x**1**)(x**0**)** |
| **−2(y**i**−θ**0**x**0**−θ**1**x**1**)(x**1**)** |

]



Is this the Python function to compute these gradient values?

**def** mse\_gradient(theta, X, y\_obs):     """Returns the gradient of the MSE on our data for the given theta"""     x0 = X.iloc[:, 0]     x1 = X.iloc[:, 1]     dth0 = np.mean(-2 **\*** (y\_obs **-** theta[0]**\***x0 **-** theta[1]**\***x1) **\*** x0)     dth1 = np.mean(-2 **\*** (y\_obs **-** theta[0]**\***x0 **-** theta[1]**\***x1) **\*** x1)     **return**np.array([dth0, dth1])

: True

*You are correct! The answer “*True*” is correct because this is the correct Python function to compute gradient values.*

Gradients are a function of the entire dataset. : True

*You are correct! The answer “*True*” is correct because a gradient’s function is applied to the entire dataset to get the gradient value.*

Consider the following function call in Python:

mse\_gradient\_batch\_only(theta,batch\_indices,X,y\_obs)

What does the constructor “batch\_indices” represent? : The list of indices of data to calculate loss

*You are correct! The answer “*The list of indices of data to calculate loss*” is correct because this is what “*batch\_indices*” represents.*

The Python function np.split() is used to split an input array into a random number of subarrays. : False

*You are correct! The answer “*False*” is correct because the function*np.split()*is used to split an input array that is provided as a constructor to a function into a number of subarrays.*

Which of the following divides the entire dataset into a batch of datasets in order to save computing time? : Stochastic gradient descent

*You are correct! The answer “*Stochastic gradient descent*” is correct because stochastic gradient descent divides the entire dataset into a batch of datasets to save computing time.*

If the batch size is 1, the quality of the gradient is minimum, but the calculation is very fast. : True

*You are correct! The answer “*True*” is correct because the batch size is a parameter that gives the ability to tradeoff the quality of the gradient approximation against the runtime to compute the gradient approximation.*

In environments with large amounts of data, it is much more common to use mini-batch gradient descent. : True

*You are correct! The answer “*True*” is correct because the cost of computing the gradient on the entire dataset is too high, resulting in very slow algorithm training times.*

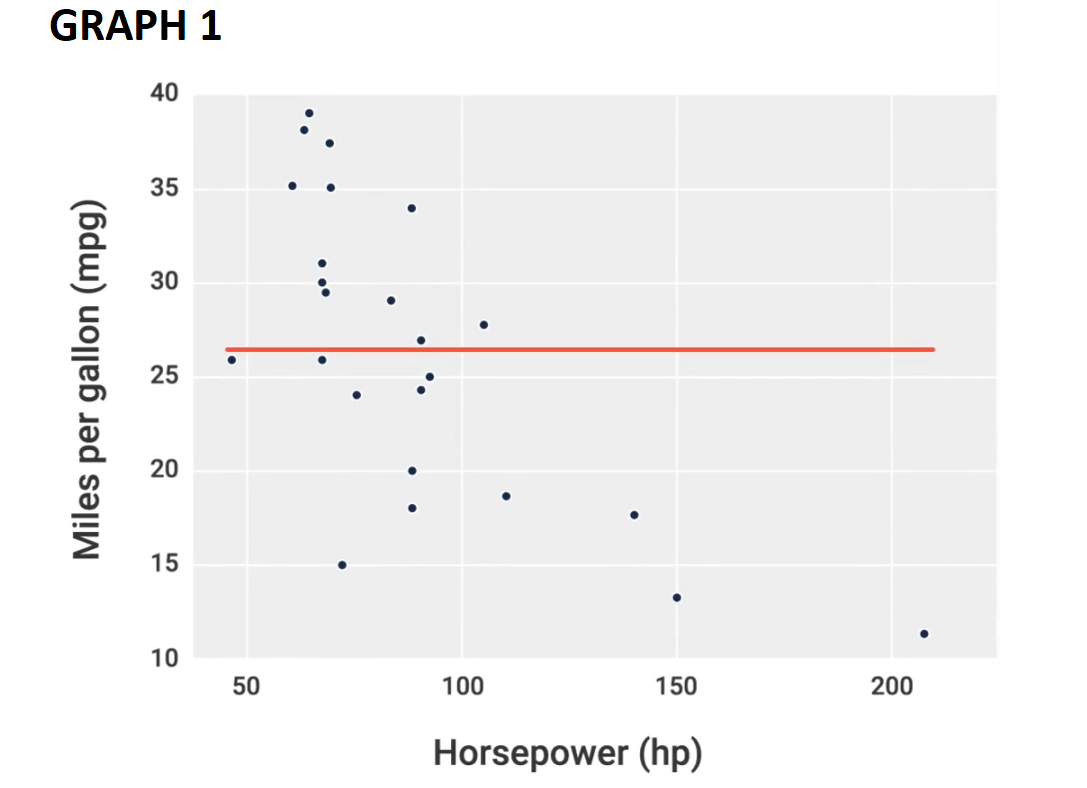
If the model is pushed to higher levels of complexity, thereby excessively increasing the parameters, the training error reduces to zero. : True

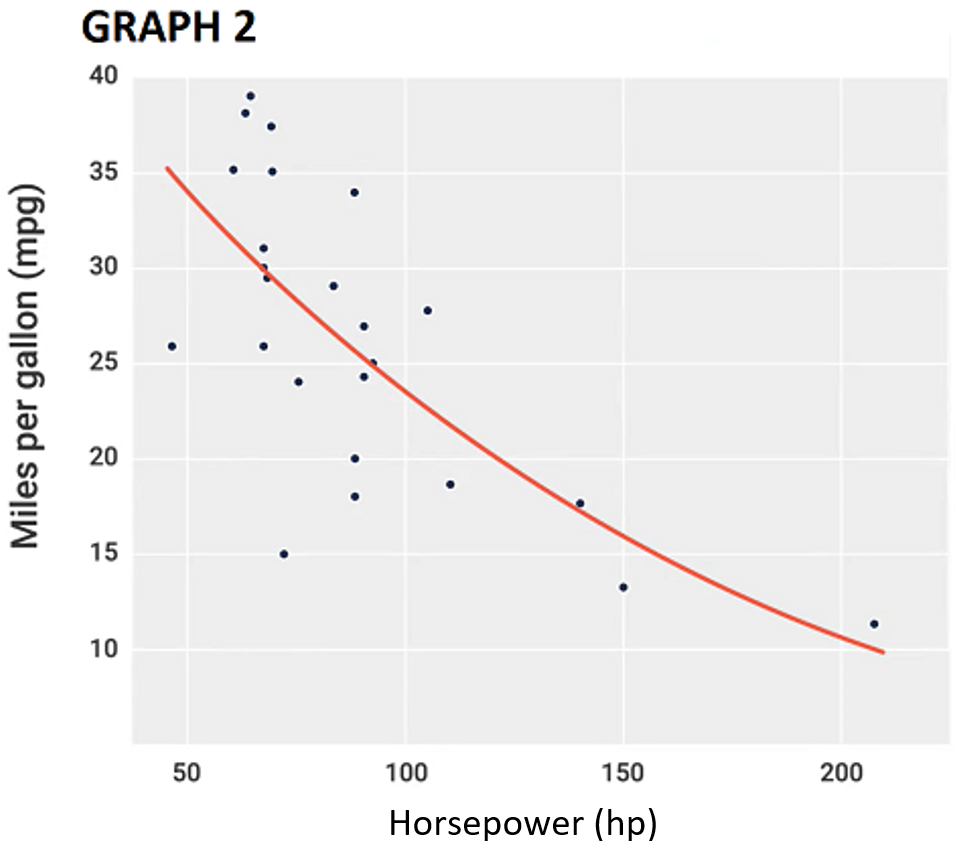
*You are correct! The answer “*True*” is correct because if the number of parameters is increased, the model is overparameterized, and there is an infinite number of valid choices of parameters that yield zero training errors.*

Overparameterized models that are trained with SGD act as if they are regularized. : True

*You are correct! The answer “*True*” is correct because overparameterized SGD results in a model that is implicitly regularized, yielding less wiggly behavior.*

Consider the following plots of two separate models:

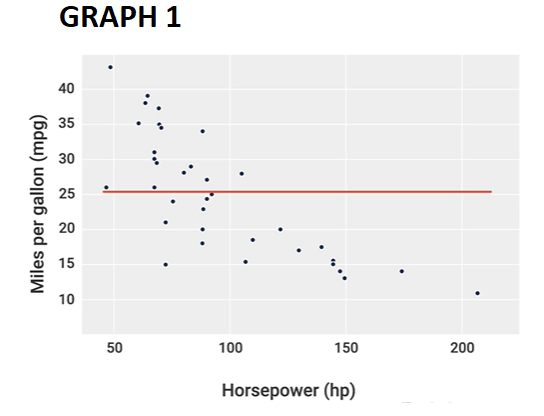


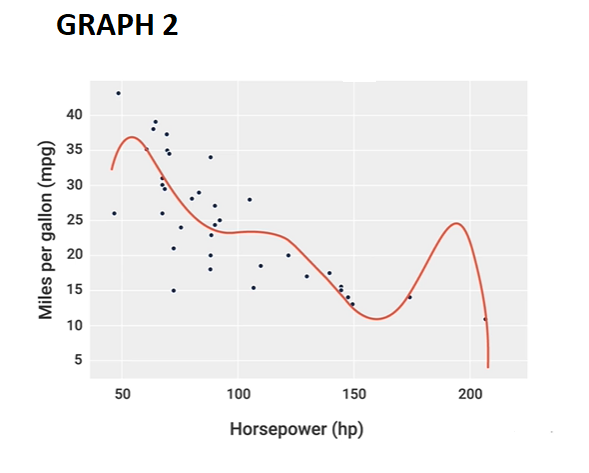


Which one has high bias? : Graph 1

*You are correct! The answer “*Graph 1*” is correct because the bias represents the fundamental inability of a model to fit the data, no matter what parameters are provided. Therefore, Graph 1 displays the inability to fit the model.*

Consider the following plots of two separate models:





Which one has high variance? : Graph 2

*You are correct! The answer “*Graph 2*” is correct because variance represents how sensitive the model is to the data. Therefore, it can be seen that Graph 2 is more sensitive to the data.*

As the model complexity increases, the bias will increase, and the variance will tend to decrease. : False

*You are correct! The answer “*False*” is correct because as the model complexity increases, the bias will decrease, and the variance will tend to increase.*

 ————— o —————