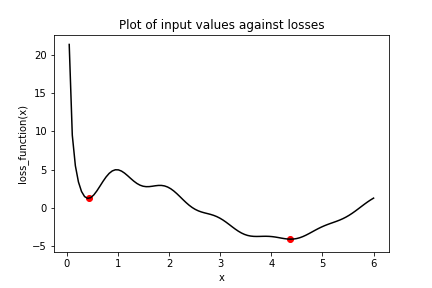
**Avoiding local minima**

The previous problem showed how easy it is to get stuck in local minima. We had a simple optimization problem in one variable and gradient descent still failed to deliver the global minimum when we had to travel through local minima first. One way to avoid this problem is to use momentum, which allows the optimizer to break through local minima. We will again use the loss function from the previous problem, which has been defined and is available for you as loss\_function().



Several optimizers in tensorflow have a momentum parameter, including SGD and RMSprop. You will make use of RMSprop in this exercise. Note that x\_1 and x\_2 have been initialized to the same value this time. Furthermore, keras.optimizers.RMSprop() has also been imported for you from tensorflow.

**Instructions**

**100 XP**

**Instructions**

**100 XP**

* Set the opt\_1 operation to use a learning rate of 0.01 and a momentum of 0.99.
* Set opt\_2 to use the root mean square propagation (RMS) optimizer with a learning rate of 0.01 and a momentum of 0.00.
* Define the minimization operation for opt\_2.
* Print x\_1 and x\_2 as numpy arrays.

# Initialize x\_1 and x\_2

x\_1 = Variable(0.05,float32)

x\_2 = Variable(0.05,float32)

# Define the optimization operation for opt\_1 and opt\_2

opt\_1 = keras.optimizers.RMSprop(learning\_rate=0.01, momentum=0.99)

opt\_2 = keras.optimizers.RMSprop(learning\_rate=0.01, momentum=0.00)

for j in range(100):

opt\_1.minimize(lambda: loss\_function(x\_1), var\_list=[x\_1])

# Define the minimization operation for opt\_2

opt\_2.minimize(lambda: loss\_function(x\_2), var\_list=[x\_2])

# Print x\_1 and x\_2 as numpy arrays

print(x\_1.numpy(), x\_2.numpy())

Good work! Recall that the global minimum is approximately 4.38. Notice that opt\_1 built momentum, bringing x\_1closer to the global minimum. To the contrary, opt\_2, which had a momentum parameter of 0.0, got stuck in the local minimum on the left.