Reverse Learning

The Input Optimization Algorithm

Gradient Descent

Gradient descent is a class of iterative algorithms that change desired parameters in the opposite (negative) direction of their gradients. The algorithm takes the partial derivative of each parameter with respect to a function and then either raises or decreases each parameter to minimize the function. Usually, this algorithm is used to change the weights, biases, or other parameters of machine learning models like artificial neural networks and linear regressions with respect to their loss functions, functions that describe the error of the model.

As an example, imagine that parabola is the graph of a loss function for a function of one parameter. The x-axis represents the value of the parameter and the y-axis is the loss with lower y-values signifying lower loss. Now imagine that we place a small ball randomly on the surface. If we imagine this system on earth, classical physics say that the ball will roll down the slope because of the pull of gravity. Similarly, gradient descent uses calculus to determine which direction is “down” and “rolls the ball” in that direction. Just like in **Figure 1,** the ball moves to a new point that now has a lower loss, and has “learned” better parameters. The algorithm repeats this step hundreds, thousands, or even millions of times until the ball is sitting at the very bottom, the minimum, where there is no more down, like in **Figure 2**. This algorithm functions the same way in systems of 3 or more variables. **Figure 3**, shows the loss function from a linear regression model, a function of two parameters.

There are many algorithms built on top of gradient descent that were designed to combat certain challenges that are faced in many scenarios. The most used variation is Stochastic Gradient Descent (SGD) which is very true to the original algorithm but deals with training data in certain ways. Most other variations are built off of SGD. The Momentum Method for SGD which increases the “speed” of the changing parameters when the gradients are steeper to combat SGD’s downfalls with steep “ravines” in the loss function. The Adagrad variation adapts the learning rate to the frequency of a given parameter, making larger steps for for infrequent parameters and taking smaller steps for frequent parameters. The Adaptive Moment Estimation (Adam) variation adapts the learning rate (similar to Adagrad) as well as adapting the learning rates (like Momentum)

Figure 1

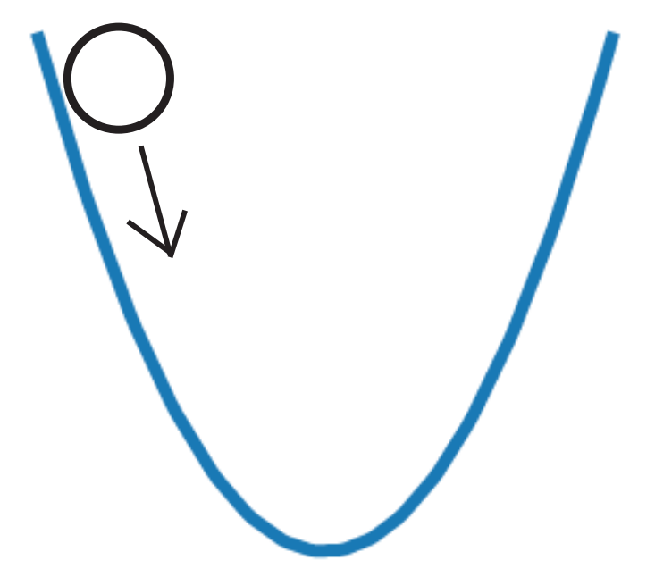


Figure 2

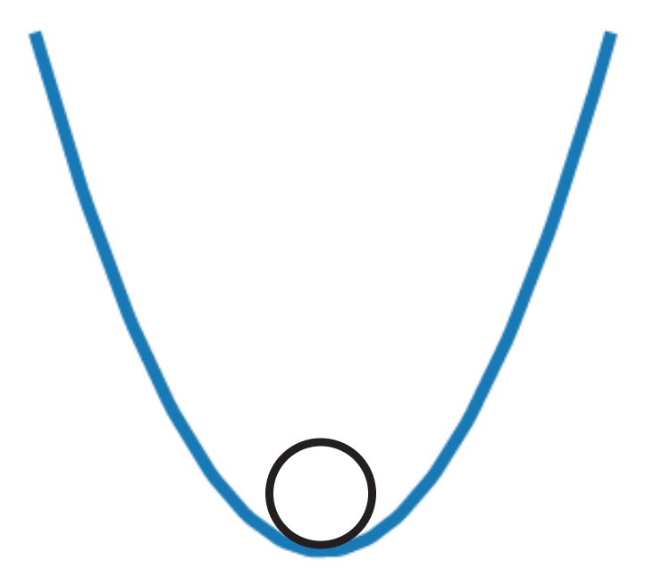
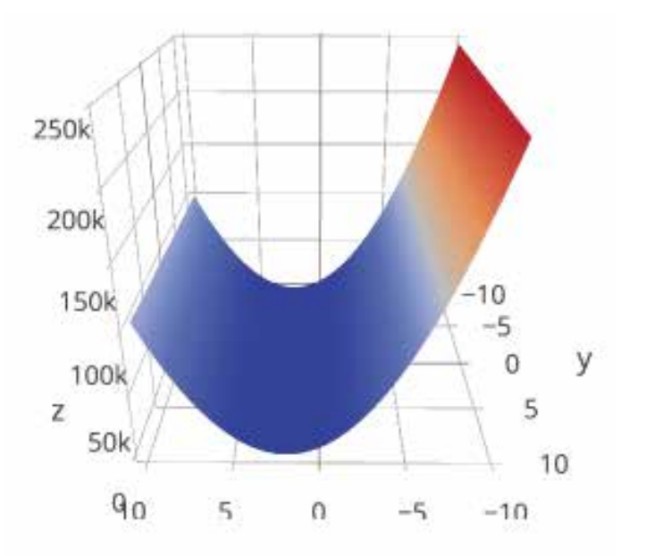


Figure 3



Problems with Gradient Descent

Gradient descent is extremely powerful but it is difficult to control. The algorithm changes each parameter only in respect to the loss function with no consideration of other factors. For example, if gradient descent were attempting to minimize the square root function, it would try to divide by zero because it didn’t consider the domain of the function. Similarly, in real-world applications, gradient descent doesn’t factor in the restrictions of the real world. Human ages can’t be less than 0 or more than 150, temperatures of less than -273.15 degrees celcius don’t physically exist, mass can’t be negative, etc. To gradient descent, values that break those conditions are perfectly possible and if they better minimize the function then they should be used, but for a researcher, any result produced that breaks a condition is useless.

Research Question

Is there a method to find meaningful-inputs that produce desired outputs on machine learning models?

Algorithm Goals

Find at least one input that produces any desired output (provided that the desired output is in the output-space)

Find local maxima / minima (provided that local maxima / minima exist)

Restrict desired inputs to user-defined ranges

The Algorithm

The Input Optimization Algorithm (IOA) works as a top layer on gradient descent optimization or any of its variations. First, the IOA sets all model parameters constant and labels the input as variable. IOA sets constant inputs as constant so that they can be used in calculations but won't be adjusted by gradient descent. Second, IOA adds an operation on top of all range-restricted inputs, a layer between the actual value and the value used in SGD. The idea is to force any input into a predefined range and still let SGD treat it like a normal operator, like in **Figure 4**. The operation is the Range-Restriction Function (R), **Figure 5**. In the function, t is the top of the desired range, b is the bottom of the range, e is Euler’s number, and x is a given input.

The operation is now included when gradient descent calculates the gradients but SGD still applies the gradients to the original value, pre-restriction. The top and the bottom of the range are individually set ahead of time to make the final, optimal input meaningful. Through this process, the actual value that is being changed may be extremely high or extremely low. That value does not matter to the user, however, because the Range-Restrict Function produces the meaningful value. Visually, you can see in **Figure 6** what a restricted area looks like to gradient descent. The same minima, maxima, and points exist, the only difference is that IOA is restricted only to the red area.

Next, IOA declares its loss function, an absolute loss between the calculated output and the desired output. When the absolute loss is zero, the algorithm has found an input that exactly produces the desired output. Then, IOA begins regular gradient descent to minimize the loss function, only, in place of modifying the parameters, it modifies the inputs to calculate better outputs.

The algorithm is finished when the absolute error is zero or below a given value, a maximum number of iterations has been reached, or all the gradients are zero or below a given value, meaning that a local minimum has been found for the loss function.

Figure 4

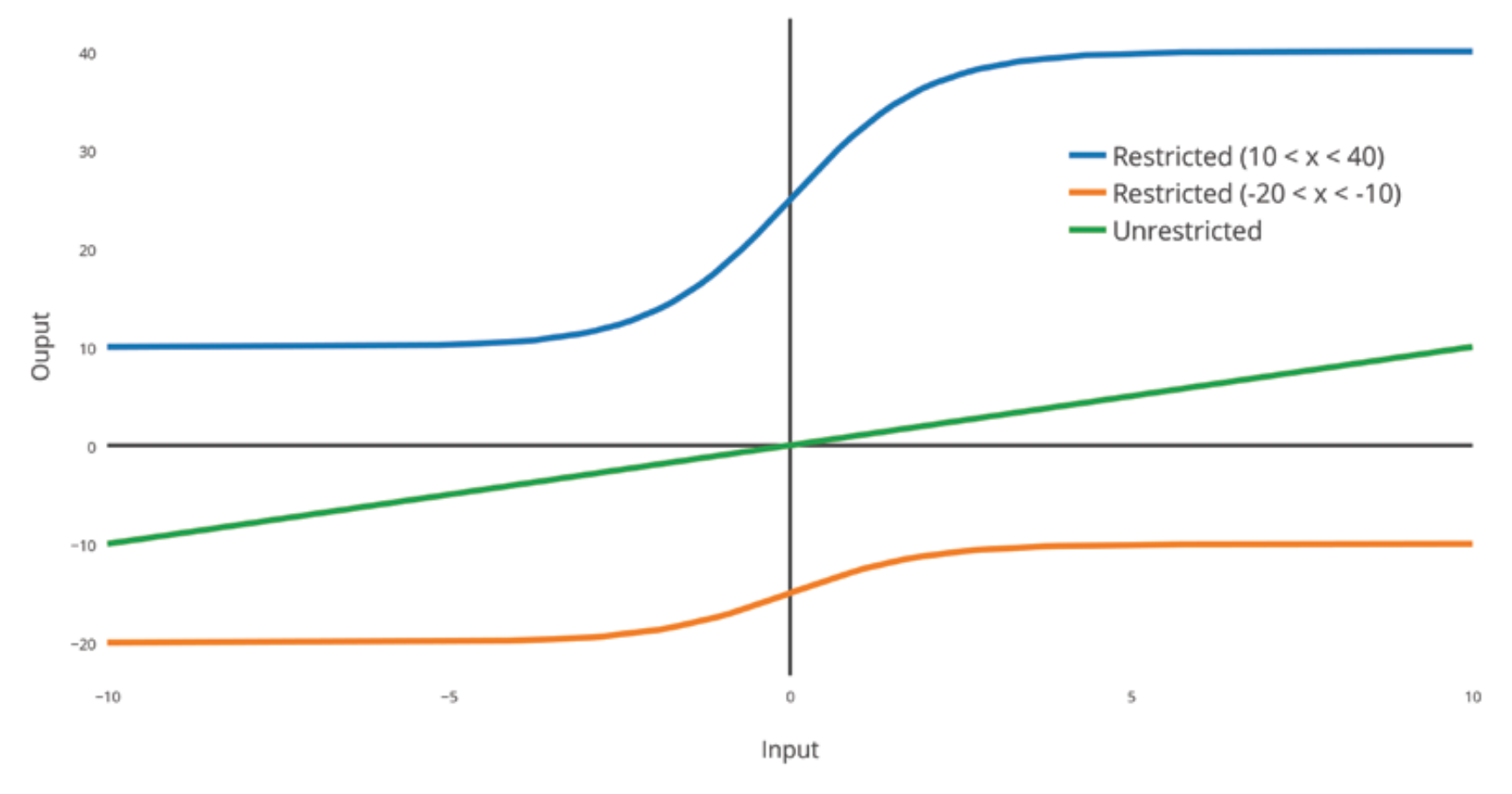
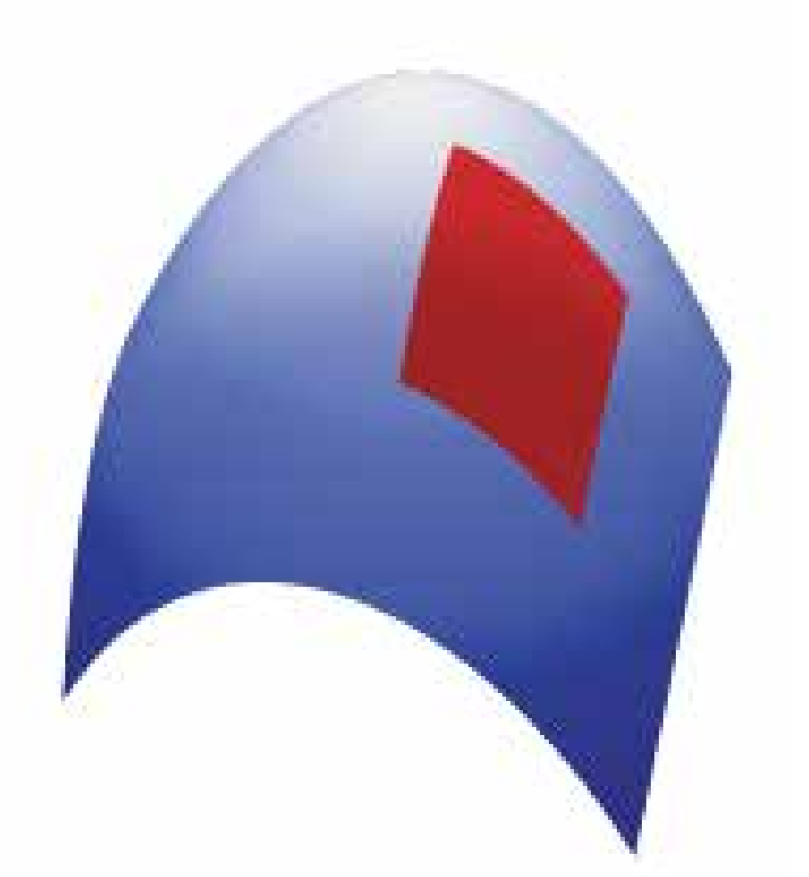


Figure 5

$$R(x) = \frac{t-b}{1+e^{-x}} + b$$

Figure 6



Changes and Improvements

The range-restriction function caused all restricted variables’ gradients to plummet. Small changes in the restricted variables caused such miniscule changes in the overall loss that the algorithm effectively discarded them and only made significant changes to the non-restricted parameters. To fix this, we simply multiplied the gradient of each restricted variable by a scalar which in turn adjusted the variables by significant amounts, enough to affect the overall function. We found that in many situations, orders of magnitude the size of 1E10 and larger were needed.

The accuracy and ability of IOA depends greatly on the accuracy of the model or function that it optimizes. In fact, IOA seems to exploit loopholes and errors in the model to achieve its minimum. At the beginning of testing, we were attempting to to use neural networks that we trained ourselves, many of which had accuracy of only 80%, error too high to obtain good results from IOA. For testing, we instead opted for using pure mathematical functions. These are, in fact, real world in machine learning as well. Functions of best fit can be determined using current machine learning methods and then fed to IOA for best results.

Future Research

IOA still is not perfect and there are many possible studies that could yield improvements. One such would be a study into other restrictions functions. The current version of IOA uses a variation of the sigmoid function but there are many other functions that place any number into a range. Does IOA find the optimal inputs faster with the hyperbolic tangent function? Is it possible to use the sine or cosine functions for range restriction? Also, the sigmoid is extremely flat (low derivative) outside of a very small range, would it be more effective to stretch the least steep range so that less of the domain maps to the extremes and there is more of a middle area?

In our experiments, we used the GradientDescentOpimizer from TensorFlow for optimization. Does IOA work even more efficiently with a variation of SGD? Does the momentum method speed the algorithm up? Does Adagrad? Adam? What other optimization methods could IOA be built on?

Also, what exact types of models could IOA be used on. Does it apply to convolutional neural networks? Decision trees? One category of models that we have not fully inspected yet is classification. There are many powerful methods in machine learning for classifying data into discrete categories, could IOA be used on top of them? What would the results mean? Is there a practical use for this?

Resources

Fredrikson, Matt. "Model Inversion Attacks That Exploit Confidence Information and Basic Countermeasures." Nature Immunology 10.11 (2009): 1145. www.cs.cmu.edu. Web.

Tramèr, Florian. "ArXiv.org Cs ArXiv:1609.02943." [1609.02943] Stealing Machine Learning Models via Prediction APIs. N.p., 9 Sept. 2016. Web. 20 Oct. 2016.

"MNIST For ML Beginners." MNIST For ML Beginners. TensorFlow, n.d. Web. 20 Oct. 2016.

Ruder, Sebastian. "An Overview of Gradient Descent Optimization Algorithms." Sebastian Ruder. Sebastian Ruder, 17 Dec. 2016. Web. 28 Feb. 2017.

Abraham Oliver and Jadan Ercoli
Brown County High School