HW3 MAE 263F

Gradient Descent and Back Propagation for Linear and Nonlinear Fitting

Jonathan Gray
UCLA

***Abstract*—In this paper we explore the use of gradient descent to minimize a loss function for predicting linear and nonlinear fits to a set of data. The algorithms used leverage back propagation to fit key parameters associated with the linear and nonlinear approximations. Here we will study the effect of varying hyperparameters including learning rate and number of epochs on the overall accuracy and runtime for convergence. The linear approximation is fairly robust for the data provided in the homework prompt, however the minimal loss achieved using a nonlinear fit exhibits diminishing returns based on the closeness of the initial guess. Additional efforts to accelerate and improve nonlinear fit predictions were explored by developing an initialization function that cycles through guesses for appropriate starting point within a user-defined tolerance. The modified algorithm with random initialization exhibits superior accuracy and speed in the final predicted fit and underscores the importance of the closeness in any initial guess for the fitting parameters. An alternative approach is also proposed for nonlinear fitting, whereby the loss function is minimized by one parameter at a time rather than all four simultaneously, this also enables closer predictions but requires iterative simulation and added run time.**

MACHINE LEARNING has had rapid development in recent years with applications spanning from generative language models developed by OpenAI, Google, Apple, and other software development teams. It also provides new avenues into data analysis, model prediction, statistically based surrogates for modeling physical phenomena, and even hyperparameter optimization within a neural network. In the physical sciences these tools stand to make a significant impact in accelerating the maturation of modeling and analysis capabilities. In general, any activity that requires calibration between the physical world and its digital counterpart stands to benefit from algorithms such as gradient descent and back propagation. Often the biggest barrier to the robustness of machine learning applications hinges on the robustness of available data to appropriately monitor and penalize the loss of a prediction. The loss function is a key source of feedback for these algorithms to self correct. At its most basic

Digital Object Identifier 10.1109/MCE.YYYY.Doi Number

Published by the IEEE Consumer Technology Society

form we explore the use of gradient descent and back propagation for curve fitting applications of both linear and nonlinear nature.

Simple Line Fit Using Gradient Descent and Back Propagation

In its simplest form, curve fitting of target data can be achieved using a straight line. The upside to this approach is the governing equation is simple and the number of parameters is limited to just two. The downsides for many applications are obvious - you cannot expect to achieve reasonable accuracy using a linear fit for a regression with significant nonlinearity. However, it does serve as a useful and digestible demonstration of these algorithms. Any straight line on some reference xy-plane can be fully described using the following equation for a Cartesian domain:

$$y = mx + b \tag{1}$$

The fundamental equation for gradient descent applications is the loss function which takes arguments for linear curve residual and sums the square of it over N data points provided. In this description we define the $Loss$ as a function of m and b in the summation shown in equation 2:

$$Loss(m, b) = \frac{1}{N}\sum_{i=1}^{N}(y_i - (m \cdot x_i + b))^2 \tag{2}$$

In this simple form, the loss function can be defined with two lines and called from a main block iterating through some user-defined hyperparameters in an effort to converge on acceptable parameters. Here, y_i is the target $i-th$ y-coordinate of data associated with known x_i coordinate for each of N points in a target regression. The loss function as defined in our algorithm therefore takes on four parameter arguments for known or $true$ x and y data as well as the target parameters for minimizing the loss, m and b.

The essential operation for exploring the accuracy of some initial guess at m and b is the computation of the gradient for the loss function. This operation is performed as a partial derivative approximation for $Loss$ with respect to each of the target parameters, in this case m and b. Equations 3 and 4 shows the derived partial derivatives of the $Loss$ for each parameter.

$$\frac{\partial Loss}{\partial m} = -\frac{2}{N}\sum_{i=1}^{N} x_i \cdot (y_i - (m \cdot x_i + b)) \tag{3}$$

$$\frac{\partial Loss}{\partial b} = -\frac{2}{N}\sum_{i=1}^{N} (y_i - (m \cdot x_i + b)) \tag{4}$$

Following the computation of the loss, back propagation is used update each parameter, feed it back into the loss function and converge on a target within some user specified tolerance. The back propagation is achieved and controlled through the use of a learning rate hyperparameter denoted by η. Equations 5 and 6 denote the back propagation process during parameter updates.

$$m \longleftarrow m - \eta\frac{\partial Loss}{\partial m} \tag{5}$$

$$b \longleftarrow b - \eta\frac{\partial Loss}{\partial b} \tag{6}$$

The remaining work for implementing the back propagation algorithm and predicting loss hinges on heuristics of the problem involved. For a linear set of target data, the linear fit is well suited for a robust and efficient convergence on a solution. However for more complex and nonlinear targets, the convergence can be largely based on the initial guess as well as the robustness of the parameters used in correlation of the data.

Starting with a fixed random seed of 42 for initial guesses, we will now explore the variation of the hyperparameters for Epochs (number of descents) and the learning rate (η) for fitting to a curve of target data defined with a normal distribution function and a small amount of noise given by:

$$f(x) = \frac{1}{\sqrt{2\pi}}e^{\frac{-x^2}{2}} \tag{7}$$

In this case, the domain (x) is set to vary from 0 to 5 using a total of 10 points evenly spaced. The resulting plot is shown in figure 1.

At first glance, this data appears to be mostly linear, therefore a linear fit should perform well in the gradient descent. Starting with 10,000 Epochs and η = 0.001, the gradient descent was performed.

The parameter correlation completed in less than 1s and the parameter prediction exhibited a loss of less than 4e-6. Increasing the number of Epochs an order of magnitude to 100,000 results in the same total loss but in twice the time (2s). Reducing the learning rate by a factor of 10 without increasing the number of Epochs results in roughly 10x the loss for 1s of runtime which visually appears to be an unacceptable amount of error for a linear fit.

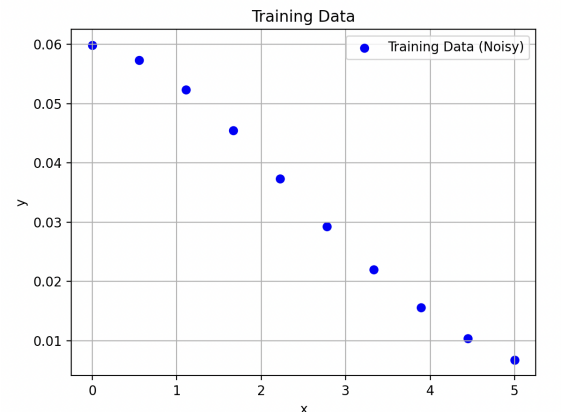

FIGURE 1. Target Data Generated

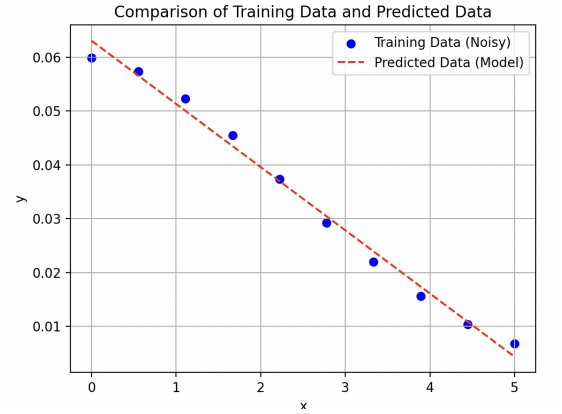

FIGURE 2. Prediction using 10,000 Epochs and η = 0.001

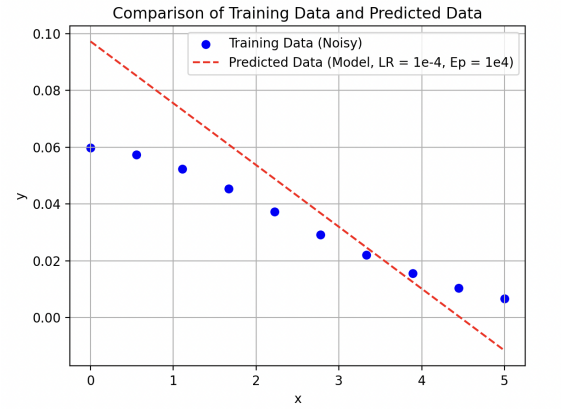

FIGURE 3. Prediction using 10,000 Epochs and η = 0.0001

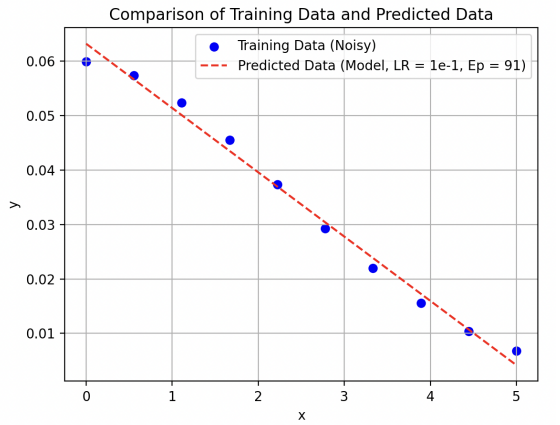

FIGURE 4. Prediction using 91 Epochs and η = 0.1

When the Epochs are increased 10x and the η is decreased by a factor of 10 we see diminished returns - the model takes twice as long as the original run to converge to roughly the same amount of loss, just under 4e-6.

Decreasing Epochs to 91 and increasing η to 1e-1 results in virtually the same prediction as the original run but with roughly 1/100th the number of Epochs and at 100x the original learning rate.

Gradient Descent and Back Propagation Using a Nonlinear Fit

Next we modify the algorithm to work for non-linear curve fitting. In this exercise the form of the residual changes to reflect a greater number of parameters governing the nonlinear behavior.

$$Loss(n,a,m,b) = \frac{1}{N}\sum_{i=1}^{N}(y_i - n\cdot exp(-a\cdot(m\cdot x_i+b)^2))^2 \tag{8}$$

In this case, we contend with the addition of two more parameters in the nonlinear loss expression, n and a. Since the gradient descent occurs as a partial derivative across all parameters individually, we now have four expressions of the loss function that are expanded out using chaing rule to the following equations:

$$\frac{\partial Loss}{\partial n} = -\frac{2}{N}\sum_{i=1}^{N}(y_i - n\cdot\exp(-a\cdot(m\cdot x_i+b)^2))\cdot\exp(-a\cdot(m\cdot x_i+b)^2)$$

$$\frac{\partial Loss}{\partial a} = \frac{2}{N}\sum_{i=1}^{N}(y_i - n\cdot\exp(-a\cdot(m\cdot x_i+b)^2))\cdot n\cdot\exp(-a\cdot(m\cdot x_i+b)^2)\cdot(-(m\cdot x_i+b)^2)$$

$$\frac{\partial Loss}{\partial m} = \frac{2}{N}\sum_{i=1}^{N}(y_i - n\cdot\exp(-a\cdot(m\cdot x_i+b)^2))\cdot n\cdot\exp(-a\cdot(m\cdot x_i+b)^2)\cdot(-2a\cdot(m\cdot x_i+b)\cdot x$$

$$\frac{\partial Loss}{\partial b} = \frac{2}{N}\sum_{i=1}^{N}(y_i - n\cdot\exp(-a\cdot(m\cdot x_i+b)^2))\cdot n\cdot\exp(-a\cdot(m\cdot x_i+b)^2)\cdot(-2a\cdot(m\cdot x_i+b))$$

The rest of the algorithm is the same as that used in the linear fit, this time we have four initial

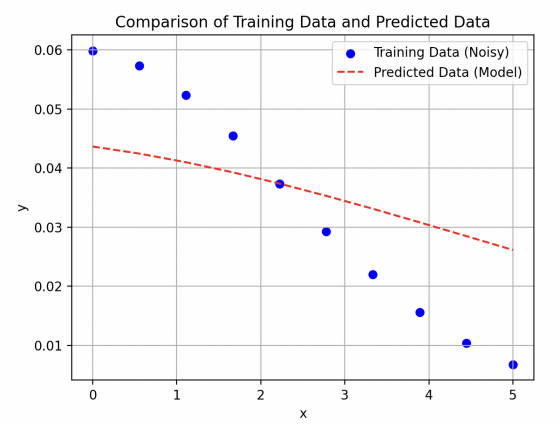

FIGURE 5. Nonlinear fit using 10,000 Epochs and η = 0.001

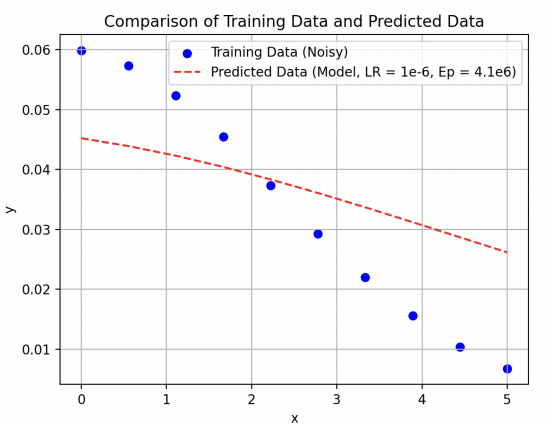

FIGURE 6. Nonlinear fit using 4.1e6 Epochs and η = 1e-6

guesses for our target parameters: n, a, m, and b. For consistency with the assignment a seed of 42 is also used for initialization of the same random guess as the hyper parameters are varied. Using the initial values provided (Epochs = 10,000 and η = 0.001) the model results in a final loss of 1.73e-4 over the course 1 seconds. The resulting fit does not look very close especially when compared to the accuracy of the linear fit.

Decreasing η by a factor of 10 and increasing the number of Epochs to account for slower learning we converge on a loss of 1.78e-4 after about 4 seconds. Increasing the Epochs to 1e7 to account for a minimal learning rate of 1e-6 gets the result down to a loss of 1.66e-4 after about 4.1e6 Epochs completed. Then around 5e6 Epochs the solution starts to diverge. This convergent value is not far off from the 1s initial set of parameters using 10,000 Epochs and η = 0.001. In fact the plots are almost identical.

If we increase the learning rate further the solution blows up - this seems to be an artifact of our algorithm and underscores the heuristic dangers of varying the hyper parameters for a nonlinear fit.

In order to explore the effect of the efficacy of the initial guess we added an initialization function that takes arguments for initialization parameters for Epochs, η, and ϵ - a user specified tolerance for initial loss to exit the initialization loop prior to the main gradient descent. Remarkably, for the nonlinear fit it is much faster to randomize a better initial guess than to try and converge on a bad one if the initialization tolerance is set to ϵ = 1e-5. Because the initialization loop discards bad guesses its more appropriate be

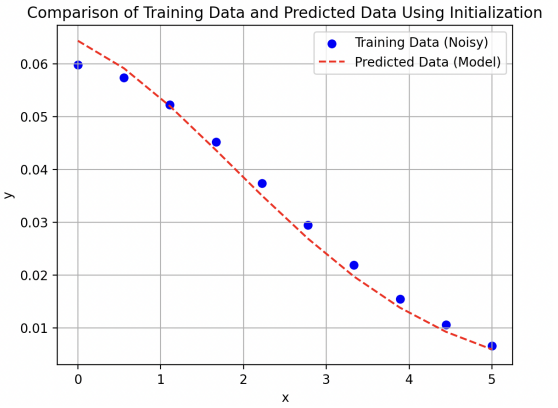

FIGURE 7. Nonlinear fit using initialization tolerance for parameter guess of 1e-5

aggressive with the initialization hyper parameters, settling with 10,000 Epochs to initialize at a learning rate of 1e-3 results in an initialization under 0.5s and a resulting loss of 5e-6 after a total of 1 total seconds of simulation.

When noise is increased, the initialization tolerance must also increase in order to maintain the speed of the initial guess. Here the noise is increased 100 fold and ϵ is set to 1e-4, without reducing this parameter the intitialization loop gets stuck for a prohibitively long amount of time. Figure 8 shows the initialized fit but with 100x the noise completed in 4s.

CONCLUSION

Gradient descent is a powerful tool for quick correlation of linear fits in target data. The nonlinear fit also provides some flexibility for a more complex scenario for correlation but the efficacy ultimately depends on

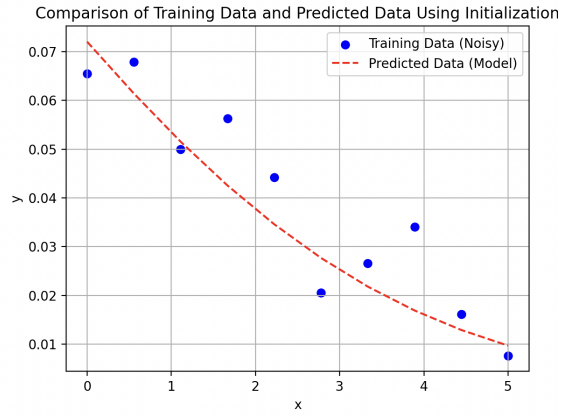

FIGURE 8. Enter Caption

the initial loss of the guess. It is therefore difficult to generate an accurate correlation at the minimum loss result by varying hyper parameters alone, in some cases it may be advantageous to update a guess quickly rather than trying to converge over a long amount of time on a bad one. An initialization function was added to first cycle through an aggressive hyperparameter definition to search for a guess that can converge within user defined tolerance before accepting the guess as the starting point for the main block. With this enhancement the accuracy of the gradient descent code increases significantly minimizing the loss by nearly 2 orders of magnitude while cutting the runtime in half. This method has also demonstrated efficacy in the presence of greater noise for a target by means of increasing the initialization tolerance parameter, ϵ.

ACKNOWLEDGMENTS

Thank you to Prof. Jawed for the resources and source material for MAE 263F including legacy code for linear and nonlinear fits.