On Approximating Functions or Fitting Models

A short essay on the topic with exercises at the bottom

Here's the situation: you have a dataset of pairs (x_i, y_i) where i is an indexing variable, assumed to be an integer and $0 \leq i \leq n$ and where n is the total number of datapoints.

In math, there are two ways to define a function g. The first way is to write an expression for $g(x)$ and the second is to write down the set total set of pairs (x, y) such that $y = g(x)$. Because the set of points for real-valued functions (functions defined on the real numbers) would be impossible to write down because there are an uncountably infinite number of such points, we prefer to define functions with the former option.

Now when we have a dataset of points, we can imagine that there is a function $F(x)$ and these points come from the set of pairs (x, y) such that $y = F(x)$. However, knowing just the datapoints is useless to us if we want to know the value of the function for some x not included in the data. Our challenge is to develop a method for figuring out what F is.

Let's take a moment to think about how we'll solve this problem. We need to reconstruct the rule version of the function F from its datapoints.

To start, we make an informed assumption about the symbolic structure of F – whether it is linear, polynomial, etc. I.e., you choose the structure of the function you think best describes the data. Some functions will be of the format $f(x) = Ce^x$, others of the form $f(x) = Ax^2 + Bx + C$, and yet others of the form $f(x) = m \ln x + b$.

This symbolic structure will be formally defined as a model:

Definition 1.1

> A **model** is a function f with parameters $w_1, ..., w_n$ and independent variable x declared as
>
> $$y = f(x, w_1, ..., w_n)$$

Once we have made an educated guess as to the structure of our model, we will want to find the best set of parameters to our model. We will call this endeavor "*fitting a model*"

Definition 1.2

> We say we are trying to **fit** a model f on a set of data points of some unknown function F $\{(x_i, y_i)\}$, if our goal is to best approximate F with f

That is, we want to know what the best parameters are so that f is as close as possible to F. But we are still groping around in the dark because our definition of "approximate" is ambiguous

Definition 1.3

> We say we are trying to approximate a function F with a model f if we are trying to find the set of parameters $\{w\}$ to f such that for any x_i in our dataset the value $\hat{y_i} = f(x_i, w_1, ..., w_n)$ is as close as possible to the actual value of the function at x_i which is $F(x) = y_i$.

What do we mean by making a model "as close as possible" to a function.

We define some function which takes in the actual value of the function and our guess and spits our a number representing how bad our guess is. Let us say that this is called a distance function defined as $D(\hat{y}, y)$. Formally, it takes as input an actual value of the function at some x value, $F(x) = y$ and a value of the function predicted by our model $f(x) = \hat{y}$, and spits out a number. This number will be how far $\hat{y}$ is from y.

Our goal will be to find the set of parameters to our model so that we get our distance e as low as possible across the entire dataset.

The most naive way to approximate F is to just keep on making random guesses for the set of parameters to the model f. So if we have q number of parameters, we just keep on find q random numbers and feed them as parameters to f and choose the set which minimizes

Of course this method can work on the first try, but it is exceedingly unlikely it actually will. We really need an algorithm for finding the set of parameters to our model.

It will have the following flavor:

1. Start with some set of parameters
2. Determine how much error there is with these parameters
3. Change each parameter an amount proportional to how much they contribute to the error
4. Keep on doing this until the error converges on some minimum.

Getting Specific

Here's where we are now:

1. We have a symbolic model which we think will be a good approximator to our function
2. We need to **fit** this model by making its guesses as good as possible.

We shall say that getting our guesses to be as good as possible means finding the parameters which make the distance between the actual value of the function and the one our model spits out as low as possible over the entire dataset. We shall denote this distance by the function D. For our purposes, we will define an error function which takes the values of D for all actual values of the function and predicted values of the function and spits out a number representing the error of our model. We will need to get this error as little as possible.

If we take D to be defined as the square of the distance between the actual value of the function and that value predicted by our model

$$D(\hat{y}, y) = (y - \hat{y})^2$$

Then we can define the error function called E. We need to get the value of E as low as possible. Taking

$$E = \frac{1}{n} \sum_{j=0}^{m} d(\hat{y}_j, y_j)$$

Which becomes, when we expand D and remember that
$\hat{y}_i = f(x_i, w_1, ..., w_n)$

$$E = \frac{1}{m} \sum_{j=0}^{j=m} (y_j - \hat{y_j})^2$$

where m is the number of data points we have for the function we are trying to approximate F.

Notice that given some input data, E depends only the parameters w_j just like we want it to.

This expression for error E essentially says that for each datapoint, take the difference of the predicted value and the actual value, square it, sum up all such datums, and then average them to the number of datums.

Now, with everything defined, we will want to find the set of parameters $\{w_i\}$ which gets the smallest error possible, trying to get the error to zero. In other words, we want to find a set of parameters for which E is at a minimum. Stated more bluntly, we want make our guesses for the output of the function on all data points closest to what it actually is.

How do we do this?

The Secret Sauce – The Learning Rule

If we know how much each parameter w_i contributes to error, we can update it accordingly. Let's say we know that the rate of change of error E as w_i increases is r_i:

$$r_i = \text{ rate of change of error for each } w_i$$

Then we can change w_i in the direction for which E decreases.

fIf we define some α satisfying the constraints $0 < \alpha < 1$, then we can use a rule for updating each parameter based on how strongly it contributes to error:

$$w_{i,t+1} = w_{i,t} - \alpha r_i$$

This is the most important equation here.

Take a moment to think about what kind of a statement it is making. We are updating the parameter w_i by an amount linearly proportional to how strongly it contributes to error. The minus sign is because r_i tells us how much the error increases as w_i gets larger, so we need to go in the opposite direction for error to get smaller with respect to w_i. Now, if w_i contributes strongly to error, it will be updated much because r_i will be large, and if its contribution is less strong, it will be updated less. This is all directly implied by the update rule.

In practice, we start the learning process by initializing each of the w_i parameters at a random value.

Note that the i's under the parameters w and rates r, are different that the $i's$ under the data points and model output $y_i, x_i, \hat{y}_i$. i is just a general indexing variable.

This is good and all, but you haven't told me how I can calculate how strongly each parameter effects error.

Bear with me here if you don't know calculus.

Definition 1.3

> Lets say we have a function with multiple independent variables $g(v_1, ..., v_j)$. The derivative of the function g with respect to a variable v_i is a number representing how the value of the function g changes keeping ever other variable constant and changing v_i.
>
> We denote the derivative with the notation $\frac{dg}{dv_i} = L$ which says the derivative of g with respect to v_i is some number L. You might also see the slightly similar notation $\frac{d}{dv_i}[g]$ used.

Think for a moment how this definition could very useful to us. In fact, it constitutes the core mathematical underpinnings of regression. Remember that we are trying to minimize the error E which is a function of the parameters w_i. Knowing how to take the derivative of error with respect to each parameter w_i is exactly what we need to find r_i and thereby update each w_i accordingly.

Recall that the error function E takes as inputs all data values y_i and x_i and all parameters w_i and w_i is the indepdendent variable.

Then taking the derivative $\frac{dE}{dw_i}$ for each w_i yields r_i:

$$r_i = \frac{dE}{dw_i}$$

Now that we know how to calculate r_i, let me restate the update rule more explicitly:

$$w_{i,t+1} = w_{i,t} - \alpha r_i = w_{i,t} - \alpha \frac{dE}{dw_i}$$

This means we will need to derive an independent learning rule for each w_i which we will do in **example 1.1**

Derivative Rules

As an integral aside (haha), here are some rules for computing derivatives which come up when we want to derive the expressions for each update . Why these rules are true is not trivial and will take many tens of pages of writing to explain fully.

Function multiplied by a constant:

> $\frac{d}{dx}\left[cg(x)\right] = c\frac{dg}{dx}$

> for some constant c

Sum of functions

> For some function G(x) defined as the sum of individual functions $g_1(x) + g_2(x) + ... + g_n(x)$, it is true that

$$\frac{dG}{dx} = \frac{dg_1}{dx} + \frac{dg_2}{dx} + ... + \frac{dg_n}{dx}$$

Square of the difference of two functions

> For some function $G(x) = (c - g(x))^2$ for some constant c and another function $g(x)$, it is true that

$$\frac{dG}{dx} = 2(c - g(x))\frac{dg}{dx}$$

That is all we need.

Now, there are two important theorems in calculus that are relevant to us. The proofs are not trivial and can be searched for online if you are interested. This will require a bit of background in calculus.

Theorem 1.1

> A function is at a minimum if the derivative of the function is zero

Theorem 1.2

> All functions assume a minimum value on any interval

From these two results of calculus, we know that our error function will have a minimum and that that minimum will be when it's derivative is zero.

Getting into Action

Example 1.1

Now how does all this mumbo jumbo theory actually play out for fitting a linear function of the form $F(x) = mx + b$. All we have to begin are a set of datapoints for this function.

Well, referencing what I said in the beginning, we need to know the symbolic structure of our linear function. I have already told you that we want to fit a linear model of the form

$$f(x, w_1, w_2) = w_1 x + w_2$$

We have thus every possible linear function in the structure of our model f and need only to find the set of ws which best approximate our function.

Now let us take the derivative of error with respect to each w_i, trusting that I know how to take derivatives:

Beginning with

$$E = \frac{1}{m} \sum_{j=0}^{m} d(y_j, \hat{y}_j)$$

Because the $\sum$ operator is linear and derivatives just work on each member of the sum, and $1/m$ is a constant, the derivative of E with respect to each term will be the sum of the constant times the derivative of each of the summation terms, from our derivative rules

Let the summation term be, replacing $\hat{y}_j$ with
$\hat{y}_j = f(x_j, w_1, ..., w_n)$

$$D(y_j, \hat{y}_j) = (y_j - w_1 x_j - w_2)^2$$

Then the derivative for each individual distance for each datapoint with respect to each parameter w_i

$$\frac{dD}{dw_i} = 2[y_j - w_1 x_j - w_2] \times \frac{d}{dw_i}(w_1 x_j + w_2)$$

And thus, using the derivative rules

$$\frac{dD}{dw_1} = 2[y_j - w_1 x_j + w_2] w_1 \frac{dD}{dw_2} = 2[y_j - w_1 x_j + w_2]$$

Finally, we integrate these ideas into the error function E to find the derivative of the error function with respect each of the the parameters w_i

$$r_1 = \frac{dE}{dw_1} = \frac{1}{m} \sum_{j}^{m} 2[y_j - w_1 x_j + w_2] w_1$$

And

$$r_2 = \frac{dE}{dw_2} = \frac{1}{m}\sum_{j}^{m} 2[y_j - w_1 x_j + w_2]$$

Recall from before that knowing r_i is a measure of how much the error changes with respect to each parameter. Therefore, we can apply our learning rule from before

$$w_{i,t+1} = w_{i,t} - \alpha \frac{dE}{dw_i}$$

If you imagine that each parameter to ch in the format $w_{i,t}$ then this statement tells us how to iterative these definitions for r_1 and r_2 we can continue to update each w, assuming we have defined α. Usually a smaller learning rate is better.

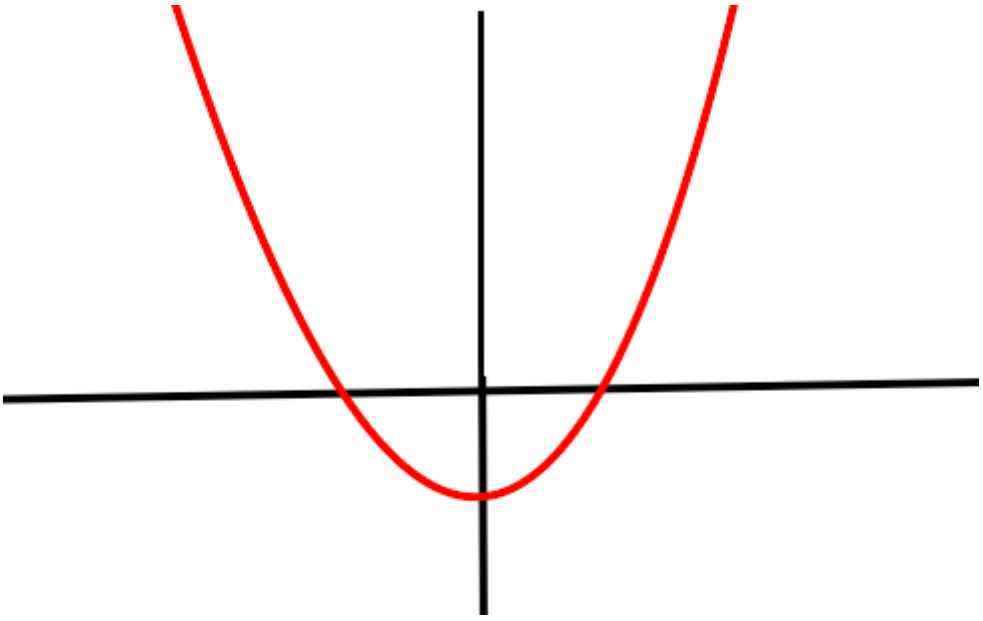
We run the update rule rule a certain number of iterations:

From **theorem 1.1** we know the error is guaranteed to reach a minimum, where $w_{i,t}$ is the error at iteration t. The values of w_1 and w_2 at the end of our iterations will result in our model f having been fit the the datapoints from the function F.

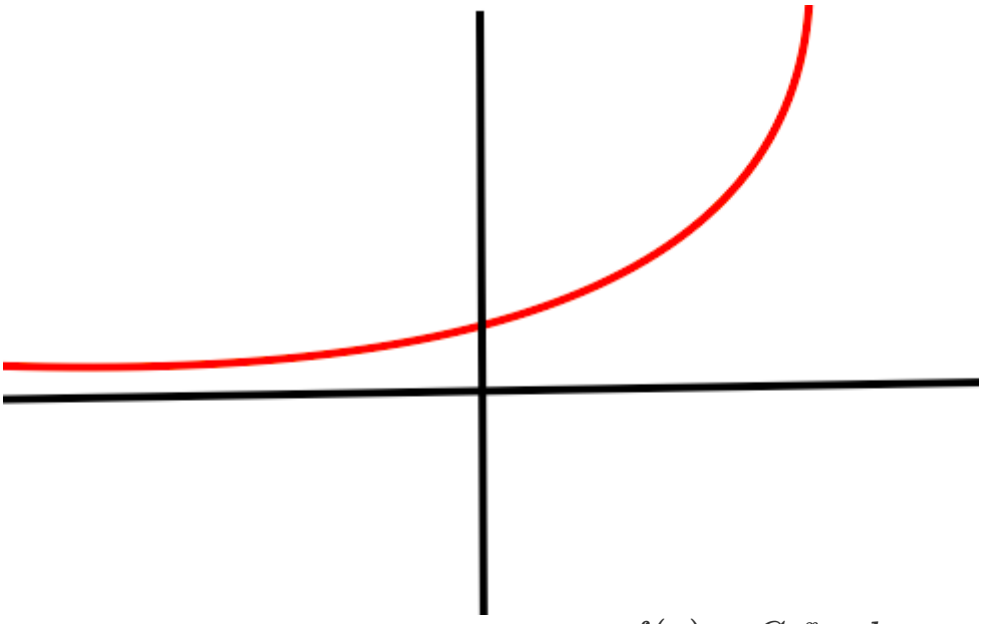
I hope this was a good introduction to the theory of numerical optimization. From here, we can start talking about more advanced theory, but this is the starting point.

Exercises

1. What symbolic form would you use to fit a model with data looking like:



2. How about for



Hint: use desmos.com to graph the function $f(x) = Ce^x + b$

3. Plot with error E on the y-axis with iteration number t on the y-axis. (a) Interpret the value of an arbitrary point (E_i, t_i) on this plot in relation to the parameters w_i. (b) Find the tangent line to the function at this point and interpret it in relation to
4. Given a model $f(x, w)$. (a) Draw an example plot of error E with respect to the one parameter w. (b) based on your plot, come up with an example of what the symbolic structure of the model is? (c) Is it possible for the model in part (a) to have two optimal solutions? Plot an example of the error function with respect to w showing what this would look like.
5. Give an example [you can simply illustrate a function] of why choosing a very high learning rate α can lead to problems in the learning process. Hint: think about the case of a very simple model $f(x_i, w)$ and how big changes to w_i can affect error.
6. Is there a pathological case when the data is barely trainable or when the model isn't actually meaningful?
7. How would you do training if the data seems to be generated by piecewise function for F (the function we are trying to fit with the mdoel f)
8. Recall $D(y, \hat{y})$ is a distance metric between our model's output and the actual value of the function we are trying to fit. (a) provide an alternative definition of the distance metric (hint: absolute difference perhaps) (b) how does it compare to our current choice for a distance metric we have already defined?
9. Why do you think the error function is defined as the mean distance $< D(y, \hat{y}) >$ and not just as the sum of distances?
10. Try implementing learning in python [or whichever language you are most familiar with] with the example model and learning rates we derived in **example 1.1**

- To do this, you need to choose arbitrary constants for the parameters to the function $F(x)$ which the model will try to approximate.
- Generate some data with these parameters by generating a set of $(x, F(x))$ pairs.
- Implement the learning equations and try to recover the values you picked for the parameters from the data alone using the model. Note: **you will need to define a number of iterations to train your model for.**
- Analyze how the model learngs by making a plot of error with respect to each iteration of the training process and a 3D plot of error with respect to each pairing of (w_1, w_2).
- From your plots of error with respect to training iteration t, what is the minimal iteration for which the model would be sufficient?
- Now try training the model another two times and making the plots again. Compare the plots. What phenomenon do you notice?
- Try using the distance metric you defined from question (8) to train this model. Compare your results with the training with the original distance metric.
- Now regenerate the data but for each datapoint add some random noise from the normal distribution (or just regular uniform random noise). Redo your plots and see whether random noise changes anything.

12. Derive an equation for r given a model with the format $f(x, w) = wx$ with the derivative rule $\frac{dD}{dw} = (y - wx)w$
13. **Implement this model in python.** using the code you wrote for example (10)

- Generate a dataset with the function $f(x) = w$ for some w you choose
- Implement learning with your answer to question 3 in python
- Run the algorithm four times for $\alpha = 0.025, 0.2, 0.6, 1.0$ for a number of iterations $n = 1000/\alpha$
- For each of the four times you run the algirithm, plot the values of w and E for every iteration.
- Compare your plots for each time you run the algorithm.
- Write up a little report summarizing your results and your thoughts as to why they are what they are.

14. Consider that you have multiple data sources which you think all are explained by the same model but differ on the placement of their x-values, i.e. for data source A the x values are between $[0, 10]$ for data source B between $[90, 130]$ etc. Can you devise a method to train one model from all these data sources?
15. This is a very important philosophical question. What is this form of function approximation good for and what is it not good for?

16. Imagine the data comes in three dimensions. How would things change? Can you try writing down equations?
17. A tip – training the inverse function is the same as training the function itself.