Chapter 6.2 - Neural Networks as Curve Fits

17 MIN · YOUTUBE · HTTPS://WWW.YOUTUBE.COM/WATCH?V=M1FOPSTONUQ
https://www.youtube.com/watch?v=m1FOpstoNuQ

SUMMARY

Neural networks can be understood as complex curve fitting models that map input data (X) to output (Y) through multiple layers of parameters (weights and biases). This process is akin to fitting a line to data but involves a significantly larger number of parameters and layers, requiring advanced optimization techniques like gradient descent and backpropagation to minimize errors effectively.

- *Neural networks are essentially sophisticated models that extend the concept of curve fitting to multiple parameters and layers.*
- *Each layer in a neural network consists of nodes connected by weights, with activation functions determining the output from one layer to the next.*
- *The training process involves optimizing weights and biases using gradient descent, which minimizes the error between predicted and actual outputs.*
- *Stochastic gradient descent (SGD) is employed to enhance efficiency by updating weights based on subsets of data, helping to avoid local minima.*
- *The architecture of a neural network can vary, with decisions on the number of layers and their sizes typically made through hyperparameter tuning.*
- *Activation functions used in neural networks must be simple to differentiate, allowing for efficient computation of gradients during training.*
- *The overall goal of training a neural network is to find the optimal parameters that yield the best fit to the data, analogous to minimizing root mean square error in simpler models.*

we're going to continue on with this concept of neural netw works as curits by kind of explicitly showing this in terms of construction of neural Nets and essentially demonstrating that you're going to use the same process we did when we did a line fit to data except for now we just have a lot more parameters than our two parameters that would characterize a line so but really it's identical in terms of philosophical concept of what we're going to do so a

neural network is going to be J basically a model that takes me from an input to an output so my data is X I'm trying to map out to Y I pick a Model F and Theta are the parameters of that model I had previously shown in the previous lecture that I could try to do a line fit so I could just say well what if F my model is just ax plus b and I have to find a and b so Theta where A

and B and I'm trying to basically map it out to the best line fit and I'm trying to find how to minimize My Fit to the data okay but now this is my curvefit this is the neural network now this is sort of what what many people are doing nowadays neural networks have sort of dominated the landscape of machine learning and AI over the

last number of years and what I want to advocate here is that

this is just a fancy line fit in some some sense it's not a line it's just functional that are there but let me walk through what it is you have your input layer so typically X this is your input it's your data and you're trying to map it out to Y remember before we were trying to map it to a line but now what I have is a much more sophisticated architecture where I go from the input to Hidden layers Z1 Z2

all the way to Z ofp and they're all connected to each other by some activation functions that we'll talk about in a moment and the idea is that X goes to maps to Z1 Z1 to Z2 there's a set of parameters or we're going to call weights Theta 1 going in this layer Theta 2 in this layer all the way through so now right away you can see the number of parameters that I have is going to be

massive I have all these weights connecting all of these nodes inside of these hidden layers to the output so in a lot curve fitting that we have traditionally done like we have a bunch of data and we try to do a line fit there's a bunch of data let's call it n and but the parameters I need to determine are actually only two I'm going just try to fit a line this is quite different you can have a large

amount of data you might even have a much larger number of parameters okay and this is a very different way of starting to thinking about our curve fitting is that this becomes quite large but the idea here is that has great expressibility power for the data that you want to do okay and so now we just have to specify essentially how are we actually going from one layer to another and this is typically done with

some kind of transfer function here which is I go from layer Z of K minus one to Z of K with some function okay we're going to talk about what these functions we're going to use are with parameters th of K and this is the function from the going from at F of K which takes me to the K layer okay so that's the idea here this is the basic building block of My Fit is some

function with some parameters that walks me from one layer to the next now the first and last layer are slightly different so I bring in X as the input goes to the first layer Z1 and from the zp layer which is the last I go to Y but again I still have these functions parameterizing that transfer from one layer to the next each of these layers has a large number of parameters and so these are all vectors or matricies

characterizing this often times what we also do is write it more explicitly in this form okay where there's what's called a weight and a bias these parameters now are broken up into two pieces a weight and a bias so you basically the weight is multiplying the actual value of these of at this layer Z of Kus one but you can add a constant to this which is called the bias which raises or lowers this so this is a more generic structure

of how we want to use in neural network training and so the idea here is I've got to find for every single layer and every single weight I've got to find the all the W K's and all the B of K which I've just wrapped up into Theta okay and so the overall architecture looks like this it's a compositional function so I start with X it goes to

the first layer f with its parameters Theta 1 so F of one is the first layer

and then that goes to the next layer which is f in the second layer with its parameters Theta 2 which then is the input for the F of three F of four all the way to the last layer F of P +1 with its paramet Theta + one which Maps me back to Y Tilda okay so that's my approximation so this is the curve fit and notice that I have all these functions I have to specify and I have

all of these weights and there's even a question of well how many functions should I have how many layers how big should the layers be these are all things that are really important considerations when we do practical machine learning and in fact typically are determined through what's called hyperparameter tuning in other words you often don't know how big how many you know how big your layer should be how many layers you should have this is determined by essentially playing around

with both size and length L layers and Heights of those layers you don't have an answer ahead of time but you do it until the work until it works well essentially you cross validate to see if you're doing well so now our objective then is just to learn the weights or if you want to think about it fit the best weights and this is going to be done through gradient descent we've already talked about this as an optimization

strategy back in the curve fitting section which is if I have a nonlinear curve fit which is exactly what we have here how do I determine these weights you compute the gradient and you start taking steps but now we're going to do gradient descent at massive scale because all these parameters there's a lot of them that we have to do and we have to update these weights which means we're going to do big scale gradient descent okay so how do we do it how do

you actually do this calculation there's two key things that are going to allow us to do neural network training one is gradient descent but a second is what's called back propagation and all back propagation is is the chain rule remember we've set up a compositional function and all the chain rule does is when you take the derivative you're trying to minimize this error as a function of these parameters so when I say you're trying to minimize the error

as a function of parameter that means you're going to take the derivative spe of this error respect to that parameter and try to set it to zero you go right back to basic calculus for this but that's going to have huge implications when this is a compositional function remember we're going through multiple layers right with all these thas to get from the input to the output and this is my output from my neural network y Tilda

and I'm trying to match it to the real data so this is a least Square fitting error so all I'm doing is the most basic objective function possible so the loss function is just root mean square error okay and so you're trying to minimize that root mean square error as a function of those weights or parameters Theta so you're going to have to figure out how to update the Theta and that's what we're going to do is

this large scale optimization is going to just use gradient descent if you remember gradient descent is you start off with some values you take the gradient of this error and you take a step downhill and the learning rate is Delta and then that gives you your new values of the weights that's how gradient descent works so so you basically take the gradient of the error function find which direction is downhill in other words towards smaller errors and you

take a step in that direction a step size of Delta that's the learning rate and then you update your weight so you do it again and again and again and again and of course this is a massive optimization problems because you might have a billion parameters that you need to find and optimize for okay luckily all this optimization is done on the back back end of tools like torch or Jacks some of these modern machine learning tools okay but it's a very

basic idea just gradient descent with a root me square error there is nothing really fancy here this is exactly what we did in curve fitting it just that our curve is this this is our curve it's some compositional function with a ton of parameters that's perfectly allowed it just that it brings you down to here which means you're going to have to think about this large scale optimization problem and in fact to make this more tractable what we're going to do is not

compute the gradient do gradient descent on all of that data we're going to do one of the key things that happens which is stochastic gradient descent in other words I'm going to do an update but now instead of all Computing the error on all the data points I'm going to pick subsets of K points so in other words I'm going to subsample okay randomly compute the gradient with a subsample of the data take a step now

there's two things it's there's two massive advantages first of all if you take let's say if my batch size that's typically what this is called is small then Computing this gradient is actually pretty tractable suppose I have a a million data points I don't have to do a million if my batch size is 100 I'm just doing gradients on 100 points instead of a million so it's really helps promote speed in the algorithm there's also but

a second huge Advantage here which is by using smaller smaller subsets of data to tell you which way downhill is you often can get unstuck from local Minima in other words the stochastic gradient descent element by randomly sampling around the data you can often instead of get caught in a local Minima the other data points will get you out of that local Minima and get you going towards the global Minima so two huge huge advantages one of them is

computational one is just getting out of local Minima but this is this was a a a com a really critically enabling step for us to be able to train neural networks is by using things like batch size and using small subsets of the data to compute the gradient because with the data sets that we have you could not compute the gradient at scale at the at for all the data okay so we're just essentially changing remember this was

just exactly like it was before but now it's subsets of data and by the way as I as I take steps I take different random subsets and once I've done all of the data that's called one Epic Epoch okay okay so you do that this is a

this is a key piece of the story of machine learning and making it tracable so let me give you a very simple example okay this is about this is a

simple neural network that I'm going to construct I have my input here it is input X it's going to go up to one hid layer which has two nodes and then come to the output and for each one of these I have F1 here F2 F3 F4 and so my output right is a linear combination of what's coming out of Z1 and Z Z2 so alpha 1 time is going to be here times the F3 which is this guy here and then what

comes over here is F4 and that's with Alpha 2 so I put the weights in front of this and this is my output here but of course F3 depends upon F1 and F4 depends on F2 so you can see the compositional structure that's there this is a very simple Network architecture but it's going to be enough for us to illustrate how gradient descent works and how the back propagation algorithm works or chain rule okay so there's my function and

these are my parameters I have the weights the I'm combining to get to the output I have you know for each one of these a two parameters A1 and B for each each one of the different functions coming through and so now if I do that then in fact what I was doing here is actually doing a line fit like's say this is parameter by two parameters a weight and a bias a weight and a bias so that's this Vector

of of the weights here that I have to find all of these variables for this very simple model I already have have you know 10 10 parameters I've got to find their best values for so the way I do that is exactly what you expect how do I minimize the error take take the derivative of the error with respect to each of the parameters let's see our A1 and then for instance in A1 I want you to show you this if I'm

taking the the output here and looking at the error if I take the derivative of the a back to A1 notice that a1's actually all the way over here so chain rule is going to take me through here and then back through here and that's exactly what we're doing here so the error which is the least square error if I come back here oops excuse me right which is right here square error then the two comes down cancels the half I

have the difference between the actual output and the data and then I've got to take the derivative in here with respect to the parameter so that's what you're seeing here Yus y Tilda then I have Dy Tilda dz1 dz1 da1 so this is just chain rule back prop and if I pick really simple functions to differentiate this going to be very easy to to actually produce a nice answer okay and so and by the way y Tilda is given by here so

I'm going to take the Der of this Vector Z1 and Z1 is given by F1 which is right here so I can compute these and so I have to compute the derivative this respect to day one so the goal and this happens in machine learning is we have to pick functions activation functions and know those the fs that I have there that I can easily differentiate in fact most the ones that are picked are trivial to differentiate and you can

just write down that in analytic form the last thing you want to do is compute derivatives numerically you want to actually just have the answer and here are some of the standard activation functions we would use there's

linear binary step Logistics 10 and Ru which is very commonly used around which is zero and then X so it's like a little zero if if your if your value of your function is negative it trims it off and

if it's positive it just it's that it's that value so these functions are all simple simple to differentiate and the reason you want Simplicity is because when I do this and I do it at scale remember I might have a billion parameters which means I've got to compute a billion of these of these directions for gradient descent and so if I can just simply have Solutions written down in other words I know what the derivatives are then that's what I

would do for this so that's the idea that's the sketch of it we're going to go into more detail later about neural NE later in the book but mostly what I wanted to highlight is you're doing the same process we did when we did a basic line fit when when you did a basic line fit we were trying to figure out how to reduce the root me square error but for us we only had two parameters we said ax

plus b how do you find a and b you took the derivative of the error respect to a took the derivative of the eror respect to B and in that case for that model we could actually work those out explicitly here what we're doing is now the same concept but a much bigger scale but we're going to compute the gradient walk downhill to update those parameters so you can see there's two key aspects here one is the optimization that you need

which is you know I just introduced creating descent second is this idea of I get to pick a curve and neural networks are a compositional structure that are as we'll see as we go forward are just sort of it's a compositional functional form but pretty simple functions underneath that which are the activation functions it's just that it's very highly parameterized and we have to update all those weights and biases in order for us to get a good fit with our neural

network and that's called the training stage of a neural network and all the training stage means is I'm doing gradient descent a bunch to walk downhill and that's called training okay so that's the idea behind this and now you have some of the basic architectures of what is actually happening in neural network fitting it just curves