01:13:47 model initialization: std 0.02, residual init

like us to be a bit more careful with the initialization and to try to follow the way gpt2 initialized their model now unfortunately the gpt2 paper and the gpt3 paper are not very explicit about initialization so we kind of have to read between the lines uh and instead of going to the paper which is quite vague um there's a bit of information in the code that open I released so when we go to the model.py we see that when they initialize their weights they are using the standard deviation of 0.02 and that's how they they so this is a normal distribution for the weights and the standard deviation is 0.02 for the bias they initialize that with zero and then when we scroll down here why is this not scrolling um the token embeddings are initialized at 0.02 and position embeddings at 0.01 for some reason so those are the initializations and we'd like to mirror that in gpt2 uh in our module here so here's a snippet of code that I sort of came up with very quickly so what's happening here is at the end of our initializer for the GPT module we're calling the apply function of NN module and that iterates all the sub modules of this module and uh applies in it weights function on them and so what's happening here is that we're in we're iterating all the modules here and if they are an nn. linear module then we're going to make sure to initialize the weight using a normal with the standard deviation of 0.02 if there's a bias in this layer we will make sure to initialize that to zero note that zero initialization for the bias is not actually the pyto default um by default the bias here is initialized with a uniform so uh that's interesting so we make sure to use zero and for the embedding we're just going to use 0.02 and um keep it the same um so we're not going to change it to 0.01 for positional because it's about the same and then if you look through our model the only other layer that requires initialization and that has parameters is the layer norm and the fighter defer initialization sets the scale in the layer Norm to be one and the offset in the layer Norm to be zero so that's exactly what we want and so we're just going to uh keep it that way and so this is the default initialization if we are following the um where is it the uh gpt2 uh source code that they released I would like to point out by the way that um typically the standard deviation here on this initialization if you follow the Javier initialization would be one of over the square root of the number of features that are incoming into this layer but if you'll notice actually 0.02 is basically consistent with that because the the model sizes inside these Transformers for gpt2 are roughly 768 1600 Etc so 1 over the square root of for example 768 gives us 0.03 if we plug in 600

1,600 we get 0.02 if we plug in three times that 0.014 Etc so basically 0.02 is roughly in the vicinity of reasonable values for the for um for these initializations anyway so so it's not uh completely crazy to be hard coding 0.02 here uh but you'd like typically uh some something that grows with the model size instead but we will keep this because that is the gpt2 initialization per their source code but we are not fully done yet on initialization because there's one more caveat here so here a mod initialization which accounts for the accumulation on the residual path with model depth is used we scale the weight of residual layers of initialization by factor of one over squ of n where n is the number of residual layers so this is what gbt2 paper says so we have not implemented that yet and uh we can do so now now I'd like to actually kind of like motivate a little bit what they mean here I think um so here's roughly what they mean if you start out with zeros in your residual stream remember that each residual stream is a is of this form where we continue adding to it X is X plus something some kind of contribution so every single block of the residual uh Network contributes some uh amount and it gets added and so what ends up happening is that the variance of the activations in the residual stream grows so here's a small example if we start at zero and then we for 100 times uh we have sort of this residual stream of of 768 uh zeros and then 100 times we add um random which is a normal distribution zero mean one standard deviation if we add to it then by the end the residual stream has grown to have standard deviation of 10 and that's just because um we're always adding um these numbers and so this scaling factor that they use here exactly compensates for that growth so if we take n and we basically um scale down every one of these contributions into the residual stream by one over theare Ro of n so 1 over theun of n is n to the 0.5 right because n the5 is the square root and then one over the square root is n.5 if we scale it in this way then we see that we actually get um one so this is a way to control the growth of of activations inside the residual stream in the forward pass and so we'd like to initialize in the same way where these weights that are at the end of each block so this C uh layer uh the gbt paper proposes to scale down those weights by one over the square root of the number of residual layers so one crude way to implement this is the following I don't know if this is uh pyro sanctioned but it works for me is we'll do in the initialization see that s that do special nanog GPT uh scale in it is one so we're setting um kind of like a flag for this module there must be a better way in py torch right but I don't know okay so we're basically attaching this flag and trying to make sure that it doesn't conflict with anything previously and then when we come down here this STD should be 0.02 by default but then if haat um module of this thing then STD * equals um copal is not guessing correctly uh so we want one over the square root of the number of layers so um the number of residual layers here is twice times Salt out config layers and then this times .5 so we want to scale down that standard deviation and this should be um correct and Implement that I should clarify by the way that the two times number of layers comes from the fact that every single one of our layers in the Transformer actually has two blocks that add to the ridal pathway right we have the attention and then the MLP so that's where the two times comes from and the other thing to mention is that uh what's slightly awkward but we're not going to fix it is that um because we are weight sharing the wte and the LM head in this iteration of our old subm modules we're going to actually come around to that tensor twice so we're going to first initialize it as an embedding with 0.02 and then we're going to come back around it again in a linear and initialize it again using 0.02 and it's going to be 0.02 because the LM head is of course not not scaled so it's not going to come here it's just it's going to be basically initialized twice using the identical same initialization but that's okay and then scrolling over here I added uh some code here so that we have reproducibility um to set the seeds and now we should be able to python train gpt2 pi and let this running and as far as I know this is the gpt2 initialization uh in the way we've implemented it right now so this looks uh reasonable to me okay so at