01:39:38 float16, gradient scalers, bfloat16, 300ms

exercised this row and um we saw that we can crop out some of the Precision inside the operation itself but we saw that we're still memory bound we're still moving around all these floats right otherwise and we're paying that cost because of this so let's now decrease the amount of stuff that we're going to be moving around and we're going to do that by dropping down to B float 16 so we're only going to be maintaining 16 bits per float and we're going to use the B flat 16 and I'll explain in a bit uh fp16 difference and uh we're going to be in this row so when we go back to the documentation here for the a 100 um we see here the precisions that are are available and this is the original fp32 the tf32 crops out the Precision and then here in bf16 you see that it is very similar to tf32 but it's even more aggressive in cropping off

of the Precision the mantisa of this float so the important thing with B float 16 is that the exponent bits and the sign bit of course remain unchanged so if you're familiar with your float numbers and I think this should should probably be an entire video by itself the exponent sets the range that you can represent of your numbers and the Precision is how much Precision you have for your numbers and so the range of numbers is identical but we can we have fewer possibilities within that range because we are truncating the Mena so we have less Precision in that range what that means is that things are actually fairly nice because we have the original range of numbers that are representable in float but we just have less Precision for it and the difference with fp16 is that they actually touch and change the range so fp16 cannot represent the full range of fp32 it has a reduced range and that's where you start to actually run into

issues because now you need uh these gradient scalers and things like that and I'm not going to go into the detail of that in this video because that's a whole video by itself but fb16 actually historically came first that was available in the Volta series before Amper and so fp16 came first and everyone started to train in fp16 but everyone had to use all these gradient scaling operations which are kind of annoying and it's an additional source of state and complexity and the reason for that was because the exponent range was reduced in fp16 so that's the i e fp16 spec and then they came out with bf16 and the Ampere and they made it much simpler because we're just truncating manessa we have the exact same range and we do not need gradient scalers so everything is much much simpler now when we do use bf16 though we are impacting the numbers that we might be seeing in our pytorch code these this

change is not just local to the operation itself so let's see how that works um there's some documentation here that so I think this is probably the best best page to explain how to use mixed Precision in pytorch um because there are many other tutorials and so on even within pitor documentation that are a lot more confusing and so I recommend specifically this one because there's five other copies that I would not recommend and then when we come here ignore everything about everything ignore everything about gradient scalers and only look at torch. AutoCast and basically also this comes to a single line of code at the end so this is the context manager that we want and we want to use that in our Network when you click into the torch. AutoCast autocasting it has a few more uh a bit more guideline for you so it's telling you do not call B flat 16 on any of your tensors just use AutoCast and only

surround the uh forward pass of the model and the loss calculation and that's the only two things that you should be surrounding leave the backward and the optimizer step alone so that's the guidance that comes from the P team so we're going to follow that guidance and for us because the L calculation is inside of the model forward pass for us we are going to be doing this and then we don't want to be using torch Flo 16 because if we do that we need to start using gradient scalers as well so we are going to be using B float 16 this is only possible to do an ampere uh but this means that the changes are extremely minimal like basically just this one line of code um let me first break in to here before we actually run this so right after logits I'd like to show you that different from the tf32 that we saw this is actually going to impact our tensors so this Lis tensor if we now look at this and we look at the dtype we suddenly see that this

is now B float 16 uh it's not float 32 anymore so our activations have been changed the activations tensor is now B FL 16 but not everything has changed so model. Transformer wte uh this is the weight uh token embedding table it has a weight inside it and the dtype of this weight this parameter is still torch float 32 so our parameters seem to still be in float 32 but our activations the loits are now in P 16 so clearly this is why we get the mixed Precision some things pytorch is keeping inlow 32 some things pytorch is converting to lower Precision um and what gets converted at what point is not super clear I remember scrolling down is it here okay I can't find it I I thought it was here okay there we go so there are a few docks on when you're using this AutoCast what gets converted to B FL 16 and and when so for example only these Matrix multiply like operations get converted to float 16 but a lot

of operations remain in float 32 so in particular a lot of normalizations like layer norms and things like that not all of those layers might be converted um so only some layers selectively would be running B flat 16 but things like softmax uh layer Norms uh log um log soft Max so loss function calculations a lot of those things might remain in float 32 because they are more susceptible to Precision changes major multiplies are fairly um robust to Precision changes uh so some parts of the network are um impacted more or less by the Precision change um so basically only some parts of the of the model are running in reduced Precision let's take it for a spin and let's actually see what kind of improvement we achieve here okay so we used to be 333 milliseconds we're now 300 and we used to be somewhere around 50,000 tokens per second we're now at 55 so we're definitely running faster but maybe

not a lot faster and that's because there are still many many bottlenecks in our gbt2 we're just getting started but we have dropped down the precision as far as we can with my current GPU which is a100 we're using pytorch AutoCast unfortunately I don't actually exactly know what pytorch AutoCast do uh does I don't actually know exactly what's in B flat 16 what's in float 32 we could go in and we could start to scrutinize it um but these are the kinds of rules that pytorch has internally and unfortunately they don't documented very well uh so we're not going to go into that into in too much detail but for now we are training in B flow 16 we do not need a gradient scaler and the reason things are running faster is because um we are able to run tensor course in B FL 16 now that means we are in this row but uh we are also paying in Precision for this uh so um we expect slightly less accurate results with

respect to the original fp32 but empirically in many cases this is a worth it uh kind of tradeoff because it allows you to run faster and you could for example train longer and make up for the uh for that Precision decrease so um that's b46 for