00:37:02 sampling loop

ready to generate so let me paste in one more code block here um so what's happening here in this code block is we have this x which is of size B BYT right so batch by time and we're going to be in every iteration of this loop we're going to be adding a column of new indices into each one of these rows right and so these are the new indices and we're appending them to the the sequence as we're sampling so with each Loop iteration we get one more column into X and all of the operations happen in the context manager of torch. nograd this is just telling pytorch that we're not going to be calling that backward on any of this so it doesn't have to cach all the intermediate tensors it's not going to have to prepare in any way for a potential backward later and this saves a lot of space and also possibly uh some time so we get our low jits

we get the loow jits at only the last location we throw away all the other low jits uh we don't need them we only care about the last columns low jits so this is being wasteful uh but uh this is just kind of like an inefficient implementation of sampling um so it's correct but inefficient so we get the last column of loow jits pass it through soft Max to get our probabilities then here I'm doing top case sampling of 50 and I'm doing that because this is the hugging face default so just looking at the hugging face docks here of a pipeline um there's a bunch of quarks that go into hugging face and I mean it's it's kind of a lot honestly but I guess the important one that I noticed is that they're using top K by default which is 50 and what that does is that uh so that's being used here as well and what that does is basically we want to take our probabilities and we only want to keep the top 50 probabilities and anything that is lower than the 50th

probability uh we just clamp to zero and renormalize and so that way we are never sampling very rare tokens uh the tokens we're going to be sampling are always in the top 50 of most likely tokens and this helps keep the model kind of on track and it doesn't blabber on and it doesn't get lost and doesn't go off the rails as easily uh and it kind of like um sticks in the vicinity of likely tokens a lot better so this is the way to do it in pytorch and you can step through it if you like I don't think it's super insightful so I'll speed through it but roughly speaking we get this new column of of tokens we append them on x and basically The Columns of X grow until this y Loop gets tripped up and then finally we have an entire X of size um 5 by 30 in this case in this example and we can just basically print all those individual rows so I'm getting all the rows I'm getting all the tokens that were sampled and I'm using the decode function from Tik

tokenizer to get back the string which we can print and so terminal new terminal and let me python train gpt2 okay so these are the generations that we're getting hello I'm a language model not a program um new line new line Etc hello I'm a language model and one of the main things that bothers me when they create languages is how easy it becomes to create something that I me so this will just like blabber on right in all these cases now one thing you will notice is that these Generations are not the generations of hugging face here and I can't find the discrepancy to be honest and I didn't fully go through all these options but probably there's something else hiding in on addition to the top P so I'm not able to match it up but just for correctness um down here Below in the juper notebook and using the hugging face model so this is the hugging face model here I was I replicated the code and if I do this and I run that then I am

getting the same results so basically the model internals are not wrong it's just I'm not 100% sure what the pipeline does in hugging face and that's why we're not able to match them up but otherwise the code is correct and we've loaded all the um tensors correctly so we're initializing the model correctly and everything here works so long story short uh We've Port it all the weights we initialize the gpt2 this is the exact opening gpt2 and it can generate sequences and they look sensible and now here of course we're initializing with gbt2 model weights but now we want to initialize from scratch from random numbers and we want to actually train a model that will give us sequences as good as or better than these ones in quality and so that's what we turn to next so it turns out that using the