00:45:50 lets train: data batches (B,T) logits (B,T,C)

to start training the model and for now let's just say the device makes code go fast um and let's go into how we can actually train the model so to train the model we're going to need some data set and for me the best debugging simplest data set that I like to use is the tiny Shakespeare data set um and it's available at this URL so you can W get it or you can just search tiny Shakespeare data set and so um I have in my file system as just LS input.txt so I already downloaded it and here I'm reading the data set getting the first 1,000 characters and printing the first 100 now remember that gpt2 has uh roughly a compression ratio the tokenizer has a compression ratio of rly 3 to1 so th000 characters is roughly 300 tokens here uh that will come out of this in the slice that we're currently getting so this is the first few

uh characters and uh if you want to get a few more statistics on this we can do work count on input.txt so we can see that this is uh 40,000 lines about 200,000 words in this data set and about 1 million bytes in this file and knowing that this file is only asky characters there's no crazy unic code here as far as I know and so every asky character is encoded with one bite and so this is uh the same number roughly a million characters inside this data set so that's the data set size uh by default very small and minimal data set for debugging to get us off the ground in order to tokenize this data set we're going to get Tik token encoding for gbt2 encode the data uh the first um 1,000 characters and then I'm only going to print the first 24 tokens so these are the tokens as a list of integers and if you can read gpt2 tokens you will see that 198 here you'll recognize that as the slashing character so that is a new line and then here for example

we have two new lines so that's 198 twice here uh so this is just a tokenization of the first 24 tokens so what we want to do now is we want to actually process these token sequences and feed them into a Transformer and in particular we want them we want to rearrange these tokens into this idx variable that we're going to be feeding into the Transformer so we don't want a single very long onedimensional sequence we want an entire batch where each sequence is up to uh is basically T tokens and T cannot be larger than the maximum sequence length and then we have these t uh tlong uh sequences of tokens and we have B independent examples of sequences so how can we create a b BYT tensor that we can feed into the forward out of these onedimensional sequences so here's my favorite way to to achieve this uh so if we take torch and then we create a tensor object out of this list of integers and

just the first 24 tokens my favorite way to do this is basically you do a do view of um of uh for example 4x6 which multiply to 24 and so it's just a two-dimensional rearrangement of these tokens and you'll is that when you view this onedimensional sequence as two-dimensional 4x6 here the first six uh tokens uh up to here end up being the first row the next six tokens here end up being the second row and so on and so basically it's just going to stack up this the um every six tokens in this case as independent rows and it creates a batch of tokens in this case and so for example if we are token 25 in the Transformer when we feed this in and this becomes the idx this token is going to see these three tokens and it's going to try to predict that 198 comes next so in this way we are able to create this two-dimensional batch that's that's quite nice now in terms of the label that we're going to need for the Target to calculate the loss function how do

we get that well we could write some code inside the forward pass because we know that the next uh token in a sequence which is the label is just to the right of us but you'll notice that actually we for this token at the very end 13 we don't actually have the next correct token because we didn't load it so uh we actually didn't get enough information here so I'll show you my favorite way of basically getting these batches and I like to personally have not just the input to the Transformer which I like to call X but I also like to create the labels uh tensor which is of the exact same size as X but contains the targets at every single position and so here's the way that I like to do that I like to make sure that I fetch plus one uh token because we need the ground Truth for the very last token uh for 13 and then when we're creating the input we take everything up to the last token not including and view it as 4x6 and when we're creating targets we

do the buffer but starting at index one not index zero so we're skipping the first element and we view it in the exact same size and then when I print this here's what happens where we see that basically as an example for this token 25 its Target was 198 and that's now just stored at the exact same position in the Target tensor which is 198 and also this last token 13 now has its label which is 198 and that's just because we loaded this plus one here so basically this is the way I like to do it you take long sequences you uh view them in two-dimensional terms so that you get batch of time and then we make sure to load one additional token so we basically load a buffer of tokens of B * t+ one and then we sort of offset things and view them and then we have two tensors one of them is the input to the Transformer and the other exactly is the labels and so let's now reorganize this code and um create a very

simple data loader object that tries to basically load these tokens and um feed them to the Transformer and calculate the loss okay so I reshuffled the code here uh accordingly so as you can see here I'm temporarily overwriting U to run a CPU and importing TI token and all of this should look familiar we're loading a th000 characters I'm setting BT to just be 4 and 32 right now just because we're debugging we just want to have a single batch that's very small and all of this should now look familiar and follows what we did on the right and then here we get the we create the model and get the lits and so so here as you see I already ran this only runs in a few seconds but because we have a batch of uh 4X 32 our lits are now of size 4X 32x 50257 so those are the lit for what comes next at every position and now we have the labels which are stored in y so now is the time to calculate the loss and then do the backward pass and then the

optimization so let's first calculate the loss

okay so to calculate the loss we're