01:06:14 parameter sharing wte and lm_head

fix a bug that we have in our code um it's not a major bug but it is a bug with respect to how gpt2 training uh should happen um so the buck is the following we were not being careful enough when we were loading the weights from hugging face and we actually missed a little detail so if we come here notice that um the shape of these two tensors is the same so this one here is the token embedding at the bottom of the Transformer right so and this one here is the language modeling head at the top of the Transformer and both of these are basically two-dimensional tensors and they shape is identical so here the first one is the output embedding the token embedding and the second one is this linear layer at the very top the classifier layer both of them are of shape 50257 X 768 um this one here is giving us our token embeddings at the

bottom and this one here is taking the 768 channels of the Transformer and trying to upscale that to 50, 257 to get the Lis for the next token so they're both the same shape but more than that actually if you look at um comparing their elements um in pytorch this is an element wise equality so then we use do all and we see that every single element is identical and more than that we see that if we actually look at the data pointer uh this is what this is a way in pytorch to get the actual pointer to the uh data and the storage we see that actually the pointer is identical so not only are these two separate tensors that happen to have the same shape and elements they're actually pointing to the identical tensor so what's happening here is that this is a common weight tying scheme uh that actually comes from the original um from the original attention is all you need paper and actually even the reference before it so if we come

here um eddings and softmax in the attention is all you need paper they mentioned that in our model we shared the same weight Matrix between the two embedding layers and the pre softmax linear transformation similar to 30 um so this is an awkward way to phrase that these two are shared and they're tied and they're the same Matrix and the 30 reference is this paper um so this came out in 2017 and you can read the full paper but basically it argues for this weight tying scheme and I think intuitively the idea for why you might want to do this comes from from this paragraph here and basically you you can observe that um you actually want these two matrices to behave similar in the following sense if two tokens are very similar semantically like maybe one of them is all lowercase and the other one is all uppercase or it's the same token in a different language or something like that if

you have similarity between two tokens presumably you would expect that they are uh nearby in the token embedding space but in the exact same way you'd expect that if you have two tokens that are similar semantically you'd expect them to get the same probabilities at the output of a transformer because they are semantically similar and so both positions in the Transformer at the very bottom and at the top have this property that similar tokens should have similar embeddings or similar weights and so this is what motivates their exploration here and they they kind of you know I don't want to go through the entire paper and and uh you can go through it but this is what they observe they also observe that if you look at the output embeddings they also behave like word embeddings um if you um if you just kind of try to use those weights as word embeddings um so they kind of observe this similarity they try to tie

them and they observe that they can get much better performance in that way and so this was adopted and the attention is all need paper and then it was used again in gpt2 as well so I couldn't find it in the Transformers implementation I'm not sure where they tie those embeddings but I can find it in the original gpt2 code U introduced by open aai so this is um openai gpt2 Source model and here where they are forwarding this model and this is in tensorflow but uh that's okay we see that they get the wte token embeddings and then here is the incoder of the token embeddings and the position and then here at the bottom they Ed the WT again to do the lits so when they get the loits it's a math Mo of uh this output from the Transformer and the wte tensor is reused um and so the wte tensor basically is used twice on the bottom of the Transformer and on the top of the Transformer and in the backward pass

we'll get gradients contributions from both branches right and these gradients will add up um on the wte tensor um so we'll get a contribution from the classifier list and then at the very end of the Transformer we'll get a contribution at the at the bottom of it float floating again into the wte uh tensor so we want to we are currently not sharing WT and our code but we want to do that um so weight sharing scheme um and one way to do this let's see if goil gets it oh it does okay uh so this is one way to do it uh basically relatively straightforward what we're doing here is we're taking the wte do weight and we're simply uh redirecting it to point to the LM head so um this basically copies the data pointer right it copies the reference and now the wte weight becomes orphaned uh the old value of it and uh pytorch will clean it up python will clean it up and so we are only left with a single tensor and it's going to be used twice in the forward pass and uh this

is to my knowledge all that's required so we should be able to use this and this should probably train uh we're just going to basically be using this exact same sensor twice and um we weren't being careful with tracking the likelihoods but uh according to the paper and according to the results you'd actually expect slightly better results doing this and in addition to that one other reason that this is very very nice for us is that this is a ton of parameters right uh what is the size here it's 768 * 50257 so This Is 40 million parameters and this is a 124 million parameter model so 40 divide 124 so this is like 30% of the parameters are being saved using this weight time scheme and so this might be one of the reasons that this is working slightly better if you're not training the model long enough because of the weight tying uh you don't have to train as many parameters and so you become more efficient um in terms of the training process

uh because you have fewer parameters and you're putting in this inductive bias that these two embeddings should share similarities between tokens so this is the way time scheme and we've saved a ton of parameters and we expect our model to work slightly better because of the scheme okay next I would