00:03:39 exploring the GPT-2 (124M) OpenAI checkpoint

would like to do is actually start at the end or at the Target so in other words let's load the GPT to 124 M model as it was released by openi and maybe take it for a spin let's sample some tokens from it now the issue with that is when you go into the code base of gpt2 and you go into the source and you click in on the model. pi you'll realize that actually this is using tensorflow so the original gpt2 code here was written in tensor flow which is um you know not let's just say not used as much anymore um so we'd like to use pytorch uh because it's a lot friendlier easier and I just personally like a lot more the problem with that is the initial code is intenser flow we'd like to use pytorch so instead uh to get the target we're going to use the hugging face Transformers um code which I like a lot more so when you go into the Transformers source Transformers models gpt2 modeling gpt2 Pi you will see that they have the gpt2 implementation of that Transformer here in this file um and it's like medium readable but not fully readable um but what it does is it did all the work of converting all those weights uh from tensor flow to pytorch Friendly and so it's much easier to load and work with so in particular we can look at the gpt2 um model here and we can load it using hugging face Transformers so swinging over this is what that looks like from Transformers import the DP GT2 LM head model and then from pre-train gpt2 uh now one awkward thing about this is that when you do gpt2 as the model that we're loading this actually is the 124 million parameter model if you want the actual the gpt2 the 1.5 billion then you actually want to do- XL so this is the 12 4 M our Target now what we're doing is when we actually get this we're initializing the uh pytorch NN module as defined here in this class from it I want to get just the state dict which is just a raw tensors so we just have um the tensors of that file and by the way here this is a jupyter notebook uh but this is jupyter notebook running inside vs code uh so I like to work with it all in a single sort of interface so I like to use vs code so this is the jupyter notebook extension inside the es code so when we get the state dick this is just a dict so we can print the key and the value which is the tensor and let's just look at the shapes so these are sort of the uh different parameters inside the gbt2 model and their shape so the W weight for token embedding is of size 50257 by 768 where this is coming from is that we have 50257 tokens in the gpt2 vocabulary um and the tokens by the way these are exactly the tokens that we spoken about in the previous video on my tokenization Series so the previous videos just before this I go into a ton of detail on tokenization gpt2 tokenizer happens to have this many tokens for each token we have a 768 dimensional embedding that is the distributed representation that stands in for that token so each token is a little string piece and then the 768 numbers are the vector that represents that token and so this is just our lookup table for tokens and then here we have the lookup table for the positions so because gbt2 has a maximum sequence length of 1024 we have up to 1,24 positions that each token can be attending to in the past and every one of those positions in gpd2 has a fixed Vector of 768 that is learned by optimization um and so this is the position embedding and the token embedding um and then everything here is just the other weights and biases and everything else of this Transformer so when you just take for example the positional embeddings and flatten it out and take just the 20 elements you can see that these are just the parameters these are weights floats just we can take and we can plot them so these are the position embeddings and we get something like this and you can see that this has structure and it has structure because what we what we have here really is every Row in this visualization is a different position a fixed absolute position in um the range from 0 to 1024 and each row here is the representation of that position and so it has structure because these positional embeddings end up learning these sinusoids and cosiness um that sort of like represent each of these positions and uh each row here stands in for that position and is processed by the Transformer to recover all the relative positions and uh sort of realize which token is where and um attend to them depending on their position not just their content so when we actually just look into an individual column inside these and I just grabbed three random columns you'll see that for example here we are focusing on every every single um Channel and we're looking at what that channel is doing as a function of uh position from one from Z to 1223 really and we can see that some of these channels basically like respond more or less to different parts of the position Spectrum so this green channel uh really likes to fire for everything after 200 uh up to 800 but not less a lot less and has a sharp drop off here near zero so who knows what these embeddings are doing and why they are the way they are you can tell for example that because they're a bit more Jagged and they're kind of noisy you can tell that this model was not fully trained and the more trained this model was the more you would expect to smooth this out and so this is telling you that this is a little bit of an undertrained model um but in principle actually these curves don't even have to be smooth this should just be totally random noise and in fact in the beginning of the optimization it is complete random noise because this position embedding table is initialized completely at random so in the beginning you have jaggedness and the fact that you end up with something smooth is already kind of impressive um that that just falls out of the optimization because in principle you shouldn't even be able to get any single graph out of this that makes sense but we actually get something that looks a little bit noisy but for the most part looks sinusoidal like um in the original Transformer um in the original Transformer paper the attention is all you need paper the positional embeddings are actually initialized and fixed if I remember correctly to sinusoids and cosiness of uh different frequencies and that's the positional coding and it's fixed but in gpt2 these are just parameters and they're trained from scratch just like any other parameter uh and that seems to work about as well and so what they do is they kind of like recover these sinusoidal like features during the optimization we can also look at any of the other matrices here so here I took the first layer of the Transformer and looking at like one of its weights and just the first block of 300 by 300 and you see some structure but like again like who knows what any of this is if you're into mechanistic interpretability you might get a real kick out of trying to figure out like what is going on what is this structure and what does this all mean but we're not going to be doing that in this video but we definitely see that there's some interesting structure and that's kind of cool what we're mostly interested in is we've loaded the weights of this model that was released by open Ai and now using the hogging face Transformers we can not just get all the raw weights but we can also get the um what they call Pipeline and sample from it so this is the prefix hello I'm a language model comma and then we're sampling uh 30 tokens and we getting five sequences and I ran this and this is what it produced um hell language model but what I'm really doing is making a human readable document there are other languages but those are dot dot dot so you can read through these if you like but basically these are five different completions of the same prefix from this uh gbt 2124m now uh if I go here I took this example from here and sadly even though we are fixing the seed we are getting different Generations from the snippet than what they got so presumably the code changed um but what we see though at this stage that's important is that we are getting coherent text so we've loaded the model successfully we can look at all its parameters and the keys tell us where in the model these come from and we want to actually write our own gpt2 class so that we have full understanding of what's happening there we don't want to be working with something like uh the modeling gpt2 Pi because it's just too complicated we want to write this from scratch ourselves so we're going to be implementing the GPT model here in parallel and as our first task let's load the gpt2 124 M into the class that we're going to develop here from scratch that's going to give us confidence that we can load the open ey model and therefore there's a setting of Weights that exactly is the 124 model but then of course what we're going to do is we're going to initialize the model from scratch instead and try try to train it ourselves um on a bunch of documents that we're going to get and we're going to try to surpass that model so we're going to get different weights and everything's going to look different hopefully better even um but uh we're going to have a lot of confidence that because we can load the openi model we are in the same model family and model class and we just have to ReDiscover a good setting of the weights uh but from scratch so let's now write the gbt2 model and let's load the weights and make sure that we can also generate text that looks coherent okay