(no title)

77 MIN · YOUTUBE · HTTPS://WWW.YOUTUBE.COM/WATCH?V=Q15RHEWZPQ4&LIST=PLOROMVODV4RMQXOCAZWATUHHQ-YEMBLCV&INDEX
https://www.youtube.com/watch?v=Q15rhEWZPQ4&list=PLoROMvodv4rMqXOcazWaTUHhq-yembLCV&index=9

SUMMARY

The lecture focuses on the fundamentals of scaling laws in deep learning, emphasizing their importance in optimizing model training and performance. It discusses how scaling laws provide predictive insights into the relationship between model size, data size, and performance, allowing researchers to make informed decisions about resource allocation in training large models.

- *Scaling laws serve as predictive rules to extrapolate small-scale model performance to large-scale behavior.*
- *Historical context shows that scaling laws have roots in classical machine learning theories, linking empirical sample complexities to modern practices.*
- *Data scaling laws indicate that increasing data size generally leads to improved model performance, often following polynomial relationships.*
- *The choice of model architecture and hyperparameters significantly impacts scaling, with empirical studies showing that different configurations yield different scaling behaviors.*
- *The relationship between model size and data size can be optimized using joint scaling laws, which help determine the best allocation of resources for training.*
- *Recent work highlights discrepancies in scaling predictions, emphasizing the importance of parameter counting, learning rate settings, and batch sizes in achieving accurate scaling laws.*
- *Isoflops analysis is a practical approach for optimizing model training, allowing researchers to explore trade-offs between model size and data effectively.*
- *The Chinchilla paper illustrates the evolution of scaling laws and their implications for model training, advocating for a balance between model size and data to achieve optimal performance.*

Okay, we will get started. We're now leaving well, temporarily leaving the land of systems to talk about more deep learning stuff which for you know, we're going to have two lectures on scaling law and scaling law adjacent phenomena. So today is going to be the basics. We're just going to talk about pretty basic scaling law stuff, some of the classic works, how does you know, the basic idea of scaling laws connect to maybe things you know about from machine learning 101. And then in two lectures, we're going to do the more advanced version of the scaling laws lecture where we're going to basically go into a bunch of modern open model tech reports, things like new P and other parameter initializations and this year I'm going to throw in some more stuff for optimizers. So there will be an advanced version of this that that will try to really get to the cutting edge. Today we're going to start with all the really basic stuff.

And due to sort of some scheduling constraints, next lecture will be inference. So you're kind of going to go back and forth. You're going to do scaling laws then inference and then scaling laws again. So scaling laws is really you know, kind of the following scenario in some ways or or you run into a very quickly once you run into the scenario, right? So you know, you have your very wealthy friend who has given you 10,000 B200s for a month

and he or she has asked you to build a very good open source language model, right? Like they they would like to get something out of those B200s.

And so you know, you've already put together your infra team. That's your assignment to you know, buddies and you have let's say a good pre-training data set. That's going to be the the assignment after this, but let's say you already have that. And then you're going to go and train a big model, right? But there's lots of choices in training a big model. All the assignments are the earlier lecture two stuff that we had earlier where you have questions like you know, what architecture should I use? What should my hyper parameters be? And it's very scary to run these things on a run that is basically you know, potentially millions of dollars in cost. So how do you really take this idea of scaling up very seriously? How do you make sure that your big run is actually successful?

So of these many choices, some of which you're going to just copy out of the literature, right? The stuff that I talked about earlier in the architecture lecture is really about kind of well, you know, adopted best practices and you will probably just pick some of these out of a hat. That's fine. But really let's say you're at the frontier, you would like to you know, build something that's actually better than the best available models today, you have to start really optimizing these, right? You can't just copy the choices of others and get something that's you know, better than the state of the art. So how do we do that? Well, that's going to require us to really engage with this idea of scaling.

So scaling laws have been one very powerful tool. You might also call it a paradigm. I think scaling laws, you know, if you talk to some people who are are really in these like big labs doing scaling work, it's almost kind of like a way of life. You know, they're like we really believe in the scaling laws. It's almost a belief and you'll see why scaling laws can sometimes be quite tricky objects. But really scaling laws are these simple predictive rules of how to go from small scale model performance and behavior and to try to extrapolate them up to large scale behavior, right? And so the basic naive way to to approach model training might be to say I have a big budget. I'm going to do you know, multiple training runs on these very big runs and I'm going to tune hyper parameters on those big runs, right? But that's very wasteful.

So instead what I would like to do is I would like to do all of my optimization at the small scale and have some simple rule that allows me to extrapolate small scale behaviors to large scale behaviors, right? And if the small scale large scale are sort of connected by some very simple sort of robust connections like regularities of some kind, then you would have confidence in this kind of new scaling approach to optimizing your system. And that's really what today is all going to be about. This kind of engineering view of of a scaling law.

Okay. And to sort of start with, I'm going to you know, talk a little bit about background, right? Because I think scaling laws are very interesting in that not only are they kind of the you know, newest paradigm. Like that's kind of how a lot of these models are designed and thought of. But also they connect very closely to very classical ideas, right? So if you're a person that's a machine learning theorist, right? I think actually there are some things that you will really kind of feel at home with in some ways. So they're kind of like empirical sample complexities.

And then I'll talk through the history of of scaling laws of one kind and show you that this is not something that's really new. It's something with actually quite a bit of history and and research done to it. Um So you know, I really really had to put some math into my slides. I really feel that the need to do so. But I think one of the things that I think is very interesting is that you know, for the longest time machine learning has thought about this question of how will my model perform, right?

Especially from a theory perspective, right? Like I have this model class, how good is my model going to be? And generalization bound is kind of the theorist answer to this question, right? So you have you know, your error over some finite hypothesis class and you say oh, my error is going to be at most this much worse than the models performance on my training set, right? This is you know, classic generalization bounds. You can sort of write this down for more complex classes like smooth densities or whatever.

But importantly, you know, not only does this tell you you know, some sort of performance. Notice that these bounds are often dependent on the sample size, right? And so this is kind of telling you okay, well, this is a theoretical upper bound on my loss values as a function of how my training set grows, right? And in some ways that's very closely related to this idea of what happens as I scale up sort of the amount of data that I have, right? Those are closely related questions or you might you know, believe me that that's true.

And if you try to figure out like what is the very first scaling law paper that exists, it turns out you can go back quite far. You can go back to Bell Labs and Corinna Cortes and you know, Vladimir Vapnik and these folks that were doing like very serious you know, theory work on machine learning, you know, asked this question. Well, training classifiers on a really large data set is often very expensive. We would like to avoid doing that and to do so, maybe we can just fit classifiers on a smaller sample, fit a curve to how their error rates decay and use that as a way to estimate their performance, right? That's almost literally a data scaling law back in 1993. Right. So this idea of at the very least you know, thinking about how your models performance changes as a function of data set size is just a very classical one and there's kind of a lot of work that supports this kind of idea.

I just want to connect this to some of these older ideas so you don't really think of scaling laws as this like very new purely neural thing. I think it's a idea that has a lot of roots in empirical machine learning and also connections to theoretical machine learning. And if you start to sort of look at the history of this, you know, there's a lot of people that have said, you know, isn't it the case that maybe instead of spending money on algorithm development, maybe we should just go collect more data because if we plot how the performance of our systems, you know, behave as a function of the amount of data that we have, it turns out that you know, they are all increasing or improving in predictable ways. Like Branco and Bril I think is one of the canonical sort of NLP references I think in sort of how the amount of data and and thinking about scaling is a really valid way of improving system performance, right? So there's lots of you know, ways of thinking about this.

And I think there's also work by other NLP folks like Collobert et al in 2012. This is one of our earlier works on like different kinds of functional forms for scaling laws where once again they're increasing the amount of data that they're using to train. I think in this case machine translation systems so they have blue scores. And they're you know, asking like what is the right functional form for how a models performance behaves as a function of

increased data amount and they you know, maybe surprisingly or not surprisingly end up with the same power laws which is power three and four as really we do today, right? For the rest of the lecture, you're going to keep seeing essentially these polynomial functions power three and power four as I talk about scaling laws. And you know, we kind of knew about a lot of this back in the 2010s that like these kinds of functional forms were very predictive of model behavior as we scaled them up.

And then finally, you know, I really like mentioning this paper because for whatever reason, they don't get or Hassabis et al don't get cited quite as much as they should. I think the origins of neural scaling I would kind of point back to Hassabis in 2017. Really kind of a very forward-looking work in many ways that built these scaling laws as a function of data set size for wide variety of systems like different kinds of speech recognition systems, machine translation systems, language models and they showed that you know, these things follow these like nice polynomial trends across a really large number of domains.

And you know, it was really ahead of its time because 2017 was long before some of the earlier sort of OpenAI scaling laws and so on in in the 2020s. And you know, they talked about stuff that we talk about today like emergence. Like where suddenly you know, models might show certain kinds of capabilities because accuracy is a much more discontinuous measure than losses. And really you know, scaling by compute that if we have systems that scale predictably to large training data, then really compute is going to be really important. And then finally, you know, since we're scaling by compute, well, speed and systems optimization is going to turn into accuracy. So in some ways like a lot of the things that we see today like the phenomena of like system scaling and so on, we could have really known or you know, I could have known in 2017 had I sort of read and thought about a lot of these scaling laws papers back then and thought about them more seriously. So um kind of like the to mention that paper in in every one of these scaling law lectures I give just to say, you know, a lot of these phenomena were known far before the modern era or like the modern language modeling era and that, you know, it's it's possible to have known the the regime that we're in even before seeing these large language models. I think that's very cool to to know. Um okay. Um any questions about this sort of like semi-historical part? Yeah.

It was and I'm wondering if those are just found empirically or is there some kind of like way we can justify all of this? Um you're asking about these scaling laws and where they come from. Um these are sort of pure curve fitting exercises. Um I think uh I will talk a little bit about where scaling laws can come from um and why polynomials are arguably a very natural class of scaling laws. Um but I think um there's no golden rule that these are the only ones. I will say though that um you know, in some place theory provides a very rich place where these scaling law functional forms can come from because theory is often thinking a lot about, you know, how do error rates decay as a function of various objects and they give you candidate functional forms of various kinds. Um physicists are also very good at this because they like thinking about limits like how do the limits behave and that gives you different um uh scaling law functional forms.

Cool. Okay. All right. Any other questions? Good. So now I'm going to start talking about um more modern neural language model scaling um and I'm going to start with data scaling laws because I think to me they're the most simple and natural objects. Um talking about data scaling laws also allows me to sort of walk you through

very step-by-step on kind of how we might sort of um uh guess what the functional form of a scaling law might be and why the polynomial form is a natural one. And then I'm going to start talking about, you know, hyper parameters and architectures and all sorts of other more exotic objects um that are going to be harder to justify from intuition but still turn out to have very nice uh scaling law behaviors.

Um I think one of the things that's remarkable about kind of this scaling law style analysis has been that these power law relationships seem to hold for a huge variety of different factors. Um so the basic ones that we see in pre-training and those are going to be the ones that I'm going to talk about today um you know, the the usual ones that you see are, you know, compute on the x-axis. This is log compute, sorry, on the x-axis. Log of test loss on the y-axis or log of the data set on x-axis, log of loss on y-axis, log of parameters on x-axis, log of loss once again on y-axis, right? So we can get a variety of different x-axes that you might think of as resources in the log axis all of them sort of linearize log test loss here.

Um you can see also much more exotic um uh scaling laws. You can see scaling laws that, you know, are are um basically the the equivalent to these in downstream benchmarks. In those cases you expect to see a sigmoid um functional form as a function of compute or compute-like quantities. Um there's even been, you know, many works recently um more on the forecasting side where people are doing things like putting dates as the x-axis and strangely enough, at least for these plots like the upper envelope of these curves are often linear in various forms of um capabilities. Um there's no reason that, you know, on this this bottom side the more exotic scaling laws need to be true um but it is it turns out to be the case that I guess language model performance is often much more regular as a function of scale and resources than it initially might appear.

Um so to begin with I'm going to start with data scaling laws and data scaling laws are a very simple kind of univariate relationship that we're going to think about, right? So in this world we're going to fix our model training procedure um and in general um unless I say otherwise the models are going to generally be bigger than the data set size and I'm going to increase my data set size, you know, in a regular way and I'm going to see how the error reduces, right? And what we expect out of this relationship in general it's going to be monotone-ish if we tune our hyper parameters well, right? Because we have more data more data should help us, you know, do better at our task the error should decrease. And if you're doing something like classification or even for for next token prediction you should really go from like random guessing to some sort of like entropy-like object where you can't do any better than sort of the the irreducible error of your task, right? That's the noise floor. So we're going to have some sigmoidal-like object over here. So this is generally what we expect out of our system.

So um we're going to start with an empirical observation and, you know, you can and and should, you know, make this on your own. You'll you'll make something that looks roughly like this in parts of assignment three um which is, you know, if we just have a very big model much bigger than our data set and we increase the data set size and we put that on the log axis and on the y-axis, you know, we have log test loss once again um you will find that the two um sort of uh the points that you plot will form a very clear linear trend, right?

Um I've taken this from Kaplan in 2020. This is um I'm going to refer repeatedly to this paper. This one's um OpenAI's neural scaling laws paper. Um they have a lot of lovely uh ablations and a lot of lovely data. Now um

this relationship where I've plotted a line on a log-log plot this is a scale-free or power law relationship. Um and what does that mean, right? If I have a line on a log-log plot um many of you to many of you this is probably obvious but just to really emphasize this, right? If something's linear in a log-log plot what does that mean? Well, it means that the error that I have is decaying polynomially, right? And and also it usually means that I'm very far away from my asymptote, right? Cuz once I approach my asymptote I'm going to, you know, taper off rather than have a line.

Um so let's go through an example just to show you how uh scaling law arises very naturally on a very simple estimation problem. Okay. So forget language modeling for the moment. Um this was a language modeling class but now it is a statistics class for the next one slide or actually two slides. No. What maybe three slides. Um so in this case, right? What I want to do is I want to estimate the mean uh from a bunch of data. Um and so I have an input it's drawn from a Gaussian and I'm going to estimate the mean of this object, right? I just write down the empirical mean and then you can write down what the the error is. So what's the expected error? Well, each of these guys is Gaussian and I know that the mu hat is Gaussian so I can write down the error that's going to be sigma squared over n, right? Okay. So well, if you look at this this is kind of a scaling law, right? I've written down the error, right? And the error, if I log that, is a function of log n, right? Log n being the log x-axis, right? Um so in general, you know, if you can come up with anything that is essentially uh 1 over n to the alpha plus some constant, you know, when you plot that on a log-log plot subtracting out the constant term you're going to get a scaling law, right? So so anything of this form where you have polynomial decay in errors is going to end up giving you a a scaling law. And of course, in parametric estimation like mean estimation or regression, um you of course expect a polynomial rate, right? You don't really expect anything else.

Um and, you know, if you've taken your your stats estimation courses you know that essentially all the classical models you are expected to have a 1 over n scaling, right? If you fit a regression or so on you might get d over n or or whatever else. Um and so if this is true and this is the origins of scaling laws, if we plot our our lines we should expect to see um roughly a slope of minus one, right? Cuz that's the exponent on the x, right? Y equals minus x plus c.

Um if we fit our neural scaling laws we will find that our exponents are quite different. Um these exponents uh I've I've taken these from these two on the left from Hessness. I've taken the one on the right from Kaplan. They're about negative point one, negative point three, negative point one, roughly speaking, right? Um and so what does that mean? That means this is much slower than estimating a mean or estimating like a linear regression, right? So this much much slower convergence rate. Still polynomial but much slower.

Now, this is kind of interesting. It's saying that we're getting, you know, these kind of nice polynomial rates but the exponents are not so great, right? They're negative point one. Where might you get an estimation rate like this? Well, you would get this if your model was kind of more non-parametric, right? I promise this will be the last statistics slide of this lecture. Um let's say that I want to estimate now instead of a mean an arbitrary smooth function, right? So one thing I could do is if I'm doing a regression problem, right? I have my inputs x1 through xn um let's say they're in the unit box. I have a little bit of Gaussian noise and I have an unknown function f that I would like to estimate. Um a trivial estimator for this is I can take my, you know, space I can cut it up into little

2D boxes of length n to the negative one to the fourth. Um and then what's our estimation error?

I'll have square root n boxes. Each box is going to get square root n samples. Um and the error is going to be roughly one over square root of n or something, right? Um and in general if I have a D-dimensional function I would like to estimate, you know, in this kind of very non-parametric arbitrary way my error is going to be roughly equal to n to the negative one over D, right? So my scaling is going to be y equals negative one over D times x. And so now these kinds of more flexible functions, functions that are more flexible than the linear class, now have rates in their scaling laws that are not just one, they're one over D. And so one, you know, mental sort of model that you could have is that the neural networks I was showing you in these slides are behaving kind of like non-parametric regressors in 10 dimensions, right? Like a nearest neighbor or other kinds of uh estimators in 10 dimensions. That's roughly the rate at which they learn from data. That's kind of a cool thing to be able to know just by looking at some of these scaling laws.

Um some people have argued this more strongly. Um Barri et al. and some other uh scaling law theory folks have argued that like, you know, the scaling law exponents are actually like literally telling us that the neural network is behaving like a non-parametric smoother and so on and so forth. It's an interesting set of arguments and you can sort of buy that intuition to whatever extent you'd like. I don't quite know how much I truly truly buy this argument. There's some, you know, of the evidence might be a little sketchy. It relies on estimators of intrinsic dimension.

But it's sort of an interesting thing to think about that somehow these exponents are telling us, you know, how fast are these neural networks really learning? Okay. So, actually I'll pause for the moment here. That's really the the end of the really basic data scaling law of like one, you know, distribution and you sample from it and you learn from it. Is that all kind of clear? Like the examples and so on. Good. Oh, yes, question.

You said model is always larger than data set size. Do you mean like one-to-one like number of parameters, number of tokens or what do you Yeah, that's a that's a good question. So, the question was something like, what what do I mean when I say the model is larger than the data set size? What I mean is, when the model gets small relative to the data, like you have way more data than you have model size, you enter this irreducible error regime, right? Like you basically fitted the data as well as you can for your model class and you can't do any better no matter how much data you add, right?

So, we want to, you know, when we're thinking about scaling laws, we usually want to think about this power law regime. Which is the sort of simpler case. Either that or you explicitly fit the asymptote and then you correct for it. Those are kind of the two things. So, when I say the model is larger than the data, it can be basically any amount that it's larger than the the data as long as it's in this power law regime.

Usually like, you know, 10 times bigger than the data will have you. Cool. Good. Okay. Now we are back to other data scaling laws, right? So, data scaling laws by themselves are not quite as useful as you'd like them to be. They just tell you how fast does my model learn, right? And that's kind of useful for forecasting. It's not useful for very much else. If you want to do engineering, you are probably interested in other questions like, you

know, what is my optimal data mixture? Like what is the best way for me to pick my training data for my model? Very useful engineering question. Should I repeat my data or should I, you know, just, you know, not repeat my data? Save the compute for something else.

Or, you know, maybe I should repeat some high quality stuff and, you know, not repeat the low quality stuff. Like I could I could do various other things with our prediction. Right? There's a lot of different engineering decisions I can make with data. So, how can scaling laws help us make some of those decisions? Um, and I think one of the things that's interesting is if we, you know, think a little bit about how classical models behave, right? Data scaling laws, if we think about their origins as essentially empirical versions of generalization bounds, they kind of tell us, well, you know, data set composition for many models really affect the offset of these scaling laws, but their slopes are actually determined by kind of the model class. They're not really determined by the distribution themselves. So, the slopes might remain the same and only the intercepts might change. And the intercept might actually be really interestingly shaped. So, this is a case of a toy model with a linear regression with two different things you can sample from. If you get only one kind of data, you know, it's the errors are high no matter what. You actually want a mixture of both, right? So, if you write down kind of how this intercept behaves, you actually start to get some very interesting insights about how having more data diversity is very helpful. But what does this kind of practically mean?

Well, practically speaking, what this means is you can actually fit data scaling laws on mixtures and that can help you do some data, sorry, pre-training data optimization. So, what does this mean? Let's say you have two data sources for simplicity. You have news and you have Wikipedia and you want to know how much news to use and how much Wikipedia to use. How can you determine that? Well, you can train really small models on really small amount of data and then you can basically train or you can fit a function of, you know, how different mixture levels affect your performance.

You can do this while you scale up your models little by little and you can see how the trends change and you can try to extrapolate that out the exact same way as you would extrapolate out a data scaling law and then you can ask which one will be optimal if I keep scaling it out to my full training law. Right? So, this is this is a very simple idea of essentially fitting a functional form at small amount of compute, finding the minimum and then scaling that guy out.

So, this is a data mixing laws paper. Unfortunately, I think if you talk to anyone who's done a lot of these data mixture work will tell you, reality is a lot more noisy than kind of this ideal world would suggest. And really what has, as far as I know, happened in many cases is you end up training a bunch of small models, you pick the best data mix from the small model and you just scale that guy up. No scaling law required. And data decide, if you're interested in this kind of thing, is is a nice large scale sort of empirical study of these kinds of data mixture things. And what they found was actually, if you don't fit a scaling law and instead you just pick the best data mix at the small scale, that works well.

For what it's worth, that's consistent with the argument that the intercepts differ but the slopes don't change because if the slopes don't change, the best mixture at small scale is also the best mixture at large scale. Okay.

Um, another really interesting set of questions that I'll also mention in the context of data is repetition. I think it's increasingly the case that compute is growing. The amount of data that we have is not growing. And so there's a lot of people interested in questions of what happens when you repeat data more and more times.

And there was a nice study a few years back called scaling data constrained language models where they roughly show something like up to four epochs with standard training recipes, you you just don't get hurt at all. But if you go past that point, then actually your your realized scaling law, this sort of dark curve, it's much worse than sort of the projected scaling law if you had fresh data. That's the dash line over here. And you can also once again write down a modified functional form that's generally predictive of the behavior of this scaling law under repetition.

Um, and we can take this idea even more to the extreme. This was kind of recent work with Perseus, one of our co-advised students, where we were asking like, okay, like what happens if you take this idea to the extreme and you just, you know, consider infinite amounts of compute. So, you're allowed to epoch as many times as you want, like an infinite amount of times. What is the best thing that you can get out of this system?

Well, turns out you can't just keep repeating passes over the data. You can't keep making your models bigger. Those have diminishing returns. And so you end up reaching for other things like, you know, ensembling your models to try to squeeze more and more out of your data. But really, I think the one thing that I'll point out that's really interesting is, you know, your standard data scaling law is here in the red. It's a very nice, predictable, you know, improvement in performance as you increase the amount of data. You know, we do all sorts of interventions like regularizing and adding ensembles, you get improvements in performance but the slopes look actually surprisingly similar. This is a a lesson that you'll learn once you start fitting your own scaling laws that very very often your slopes don't change. Often the interventions that you do just change the intercept of the scaling laws.

Um, one last thing that I'll mention cuz I like this paper in the context of scaling laws and scaling in general is that a lot of the phenomena we care about in data is actually very scale dependent. And one example of this is data filtering, right? So, if you or I, you know, tomorrow decide to filter some data for the models we're training, we would probably filter very aggressively, right? We would only keep the highest quality stuff because both you and I don't have very much compute, right? So, we can't train on all the internet anyway. We might as well filter down to sort of the most high quality stuff.

On the other hand, if you have a lot of compute, right? Then you want to train on more and more stuff because you're going to not want to repeat on this very high quality data. So, as you get more and more compute, your filters kind of become looser and looser and potentially you start training on stuff that's lower and lower quality, right? So, a lot of the things that we often think about statically, like things like data filtering or the quality of data, are actually much more dynamic. As you as you increase scale, you have to think about where am I going to get the rest of the data, right? Like the filters can't stay fixed as a function of the data. And certainly like the optimal filters turn out to not be fixed as a function of scale.

Okay. So, that's it for data scaling laws. This is the part that I think is most straightforward to understand. Even as kind of like a, you know, stats ML-minded person, like this stuff is all very simple and straightforward to me. You increase data, performance gets better in a very predictable way, right? I think this is uncontroversial. The fact that it does so polynomially for such a big model is still kind of cool, but nothing really crazy. So, hopefully this is, you know, reasonable to all of you that there's so much predictability when you increase the amount of data.

Oh, yes. So, on the slide for the pre-training under infinite compute, so, that like graph on the right is like linear in both the loss and data set size. Whereas like the previous example I gave was like log-log scale. I was wondering why it's linear though. Yeah, so so this is this is a I mean, bad axis. Herman and I just talked about bad axes like this. You know, this is actually doubling, right? So, it looks linear but it's actually log scale.

You fixed it now? You fixed it now? Okay. And the Y axis here, it doesn't look logged but the range of this is so small that linear logs are basically the same. If you look at the fitted functional forms here, right? These are still polynomials. So, we're fitting the same functional forms and those are the ones that are plotted here. Um this is another point. I'll get to it at the end of the lecture. Um if you only have a tiny slice of a compute range, it's very, very difficult to tell if something is scaling polynomially or if something's scaling exponentially, right? Because Taylor approximations are a thing. Everything looks linear if you kind of zoom in enough, right? Um and so, whether this is polynomial or some other functional form, you want to be always a little bit careful and skeptical.

Okay. Okay. All right. So, that was scaling laws for data. Um in many ways straightforward. Now, I want to sort of talk about more exotic forms of scaling and sort of scaling laws for for model engineering of various kinds. Um so, what we're going to do now is we're going to try to design large language models. Um you know, maybe you're a radical and you believe that maybe LSTMs are the future. Like, why can't I use a LSTM instead of a transformer? You know, I want to blow my B200 run on my LSTMs. Or, you know, maybe you're a different kind of radical and you want to train on SGD. Like, why can't I train on SGD? Um you might also wonder, like, if you have a limited amount of resources, you have different ways to to spend it, right? Like, maybe you should train models longer, maybe you should train bigger models, maybe I should go collect more data. There's all sorts of tradeoffs always at play, and scaling laws give us quantitative ways of making these tradeoffs, hopefully.

I'm going to start off with some simple hyperparameter questions. Um things like, how do I choose my architecture or optimizer? Um things that I talked about in um the second lecture. And there, I gave you the perspective of maybe you can study what other people do. Now, I'm going to try to give you a first principle answers to those questions um with a little bit of a different set of tradeoffs, right? So, you have to believe in scaling laws in this lecture, but as long as you do, you get to sort of try things on your own and find out.

Okay. The very first question we might start with is to say, are transformers really better than LSTMs? Um you know, LSTMs, they can work okay, too. Like, they will fit distributions. You know, with the advent of things like Mamba, we know that SSMs can work. So, are they better? The brute force way to answer this question is to train a big LSTM, GPT-3, or even bigger class of models. Um the scaling law way would be to train a bunch of smaller models, um and say, I'm going to train my transformer across a different variety of compute ranges,

and I'm going to train different LSTMs with different numbers of layers on, you know, once again, a variety of different compute ranges.

Now, what does this show? This shows, you know, LSTMs have, you know, definitely different intercept, maybe even different slope than the transformer. And because of this, probably don't want to pick a LSTM, right? Um this plot would justify scaling up the transformer um instead of the LSTM. And if you look at a lot of the architecture papers today, like, you go look at the Mamba paper or um the gated DeltaNet papers, they will always have a plot that looks like this, right? Where you have, you know, vanilla transformer, our really cool model, and our really cool model is usually either on top or below, because that's what would prove that your model is doing better, right?

And you certainly don't want something where your slopes are worse, because that means as you scale up larger and larger, your models will eventually do worse, right? So, this is one way of really reasoning about architectures without uh training gigantic models. Um there haven't been, I think, quite as many uh recent and good uh architecture scaling law papers. Um I think the one that is relatively recent and that I like quite a lot is is by E T and others at Google um back when they were publishing this kind of work, where they did a whole bunch of scaling studies on on T5-style models, where they sort of trained larger and larger variants of T5 models on many different architectures.

Um and I think one thing that's really cool and one of the reasons why I still bring this paper up is, you know, they kind of capture exactly the architecture changes that we implement today in our frontier models, right? So, things like, you know, performer, which is a uh um efficient attention, that doesn't scale very well. Like, we do not implement those. Um you know, the things that we implement are the gated linear unit, right? Red line better than green line throughout the scaling trends. Um switch transformer, I don't know what happened with their biggest run, but, you know, generally speaking, it's got good scaling trends.

Um and uh I forgot Oh, mixture of soft max. Actually, we don't do this anymore, but this is probably an effective intervention given the scaling trends. Um so, you kind of see that through the scaling trends, um even though a lot of these, you know, papers were operating at much, much smaller compute scales than what we operate today, um they're able to see the kinds of architecture trends that are driving uh frontier model development back then, right? And I think this is why a lot of people kind of use scaling laws as like a really almost paradigmatic way of saying, like, "Oh, if it doesn't show up in the scaling law, it's not a good intervention." Because in some ways, um a lot of the past work has captured what we see today.

Good. Okay. Um going beyond architectures, we can extend this kind of analysis to all sorts of other things that we might um want to understand. We might ask, okay, is SGD better or worse than Adam? Um if you do the scaling law analysis, this one's by Hessness, you'll see very clear trends. Um this is once again the really mysterious thing of, you know, the intercepts are different, but the slopes are very similar, right? Um if you fix the data and you fix the models, like, slopes are often very, very similar. I I'm very surprised by this every time I see it, and yet it's very true. Um it's rare to get uh different slopes, even with an intervention as big as SGD to Adam, right? Like, if you were the one training with SGD or Adam, you would you would think that's a huge

change.

Scaling trends remain roughly the same. Um in this case, this is a uh language modeling with a recurrent highway net. Um similarly, in uh lecture two, I told you about aspect ratios. Like, how do you pick your depth-to-width tradeoff? And I told you, well, you know, you can just pick some random number that's, you know, some roughly reasonable. It's like four times the reasonable multiplier. But, if you were doing this kind of scaling analysis, you could kind of know much more precisely what is happening.

So, um the first thing you could do is you could look at scaling trends as a function of layers. And you would immediately see that having extremely few layers is very, very bad for your model. Like, if you have one layer, you know, you're you're not going anywhere. That's a terrible, terrible scaling trend. Greater than one layer, actually, it's surprisingly more competitive. Although, you do see that at every compute level, more layers, at least in in this case, is is better, right? And if you you can do a much more fine-grained study, which they do do in Kaplan, where, you know, you try to identify what you might call scale-invariant quantities, like, the aspect ratio, right? The number of layers is not a scale-invariant quantity. As you make your model bigger, you do want more layers, right? But, as you make your model bigger, maybe the aspect ratio, the optimal one, should stay the same, right? This one is scale-invariant. And so, if we look at the aspect ratio, we see that at different model sizes, you know, we get basically the same minima around 100 uh D model for every layer, um or maybe a little bit less. And it does shift a little bit, right? Like, there is a shift towards um sort of uh aspect ratio that is smaller for deeper models, but for the most part, the minima stays roughly similar.

You can do similar kinds of analysis um for things like attention head dimension and others. And if you look at the Kaplan paper, um they do quite a good job at doing all of this. And this is one way that you can be confident about your scaling strategy, right? So, if your scaling strategy is, I'm going to fix my aspect ratio and I'm going to scale up, you can make plots like this and convince yourself, you know, we're probably good because our our optimum isn't shifting too much as I go to larger and larger and larger models.

One thing that's important, and this is a you know, this is a huge can of worms, and a in a sufficiently big can of worms that I will talk about it on more than one slide. It will come back. Um you know, not all parameters are created equal. And because not all parameters are created equal, your scaling laws may look good or bad depending on how you define what a parameter is. Um so, in the Kaplan paper, they made this observation that if they drew their scaling laws for for depth um with the embedding parameters, they got these like very funky-looking scaling laws.

And they said, "Oh, this is no good at all, no good at all," right? Because it's no good at all, they decided that they were going to exclude all the all the embedding parameters, and they were only going to count the non-embedding parameters, because you know, you could justify it to yourself saying, "These are the parameters that are doing computation or whatever." Um you know, we're going to see later that this is going to have non-trivial implications on the validity of some of the results. But, the point here is, scaling laws aren't kind of magic. I think, you know, Percy likes to make this point that, like, scaling laws and all these kinds of, you know, predictability across scales is engineered, right? Like, they don't happen automatically. We need to sort of pick

the right kinds of, you know, x-axes to look at. We need to make sure that the hyperparameters for these things are set right. And only under those conditions does it become possible to get predictable scaling across many orders of magnitude compute, right? So, um this is kind of what a lot of scaling law researchers do, right? They think about what the right way is to get very predictable scaling.

Now, related to this, I think are are very interesting recent papers on um things like scaling for mixture of experts. You know, now that mixture of experts is kind of, you would say, the dominant way of training large models, you know, it is quite important to think about what is the value or what is a parameter in this new world where number of parameters and the number active parameters are kind of decoupled. Um and there's a nice uh clean analysis by some folks at Apple and MIT um that show the sort of scaling trends as a function of um function of both parameters and active parameters. And kind of the cool thing that you see is as you start to train um bigger and bigger models like if you you know increase the amount of total parameters on the x-axis over here and you want to minimize your loss you're going to end up with sort of more sparse and sparse models. See how the color becomes darker and darker and darker right? Similarly here if you like the number of active parameters kind of decreases as a function of MOE sparsity and you sort of get the result that I think is somewhat intuitive here which is you know you can draw these sort of surfaces for every sort of compute amount you can ask okay I would would I rather spend that compute on active parameters or you know in how much sparsity do I want right? I can vary my sparsity without really changing the amount of compute that I'm spending.

Now given that you can ask okay what happens if I add more sort of empty parameters same active parameters but sort of more total parameters. As you move from sort of back on the plot to in front of the front of the plot you see kind of the losses are improving right? So in other words this is saying that the parameters in MOE that are not active are still helping you reduce your loss which is kind of cool and you can kind of write down what the functional forms of this are. I won't go into to that part in more detail but you know the point here is to say all of these kinds of quantities that we care about in optimizing a MOE are things that have sort of predictable scaling and regularities.

Okay. And the two things I want to now talk about are actually really yeah two things that I now want to talk about. When you're going to train a new big model actually most of the things that I talk about are fixed right? You're you're probably all not radical enough that you're going to switch to a LSTM you're probably going to pick like I don't know a deep seek for inspired MOE or something right? And you're probably going to pick architecture choices that are tried and tested. You're not going to really throw everything out. But two things that you really do have to maybe sort of reinvent in some sense is the batch size and the learning rate right? If you train your deep learning models you know that you have to you know be very about the combination of your batch size and learning rate right? You change one you have to change the other.

Now batch size we know from the previous lecture is very tricky for another reason. We want the batch size to be as big as possible because a big batch size gives us opportunity for parallelization right? Remember data parallel requires large batch size. So now having said all that what we really want to know is how large can we make our batch size before we start to suffer right? That's the relevant systems question. And you also want to

know how does that change as a function of model size? Okay.

So given that preface there is this you know very important idea that is used in a variety of papers and talked about very frequently called the critical batch size. The critical batch size is roughly this idea that up until a certain point when you start to get diminishing returns there is returns perfect returns to increase batch size right? So I might call this the noise limited regime. So in the noise limited regime every additional element that you throw in your batch reduces the gradient noise in your SGD step right? And the since your variance limited that reduction in variance is very very helpful right?

That that gives you big returns. So this is the regime in which you have perfect scaling. You're doing about as well as you can given the amount of extra examples that you're processing. As you sort of get to a certain point where you've reached kind of the noise scale of your gradients you get to a point where you are no longer variance limited. You're now bias limited. What does that mean? Well remember in gradient descent you're looking at kind of the local structure of your objective you you don't have global sort of view of where sort of the minimum is right?

So no matter how low noise I am I'm only going to point in this light blue direction over here right? And there's always a disagreement between my local descent direction and kind of where the global optimum is. Because of that at a certain point I'm going to get diminishing returns because now my limiting factor is no longer variance it is kind of this bias term right? So now I'm bias limited past this point over here. And so critical batch is kind of a rule of thumb you might call it or a convenient trade-off point where you sort of say this is the point at which we're starting to cross over from our perfect scaling regime to our ineffective scaling regime. And so it gives you a good rule of thumb of where to be setting your batch size. If you want your batch size to be as big as possible without suffering these like huge efficiency losses. Okay hopefully that idea is very clear.

Now critical batch size is kind of a strange object it is derived from a variety of of sort of complicated calculations about you know the local quadratic approximations to the objective. But I will maybe give you the mechanical steps of what is actually used to estimate this quantity. Um So what you do is first you pick a target loss. You know target loss might be some number that you want to hit. So your goal is to hit this target loss as quickly as possible right? That's my goal right now.

What I'm going to do is I'm going to sweep over all the different batch sizes or number of different batch sizes and I'm going to write down the number of steps that I needed to reach my target loss and the number of examples that were needed to get to that target loss. Of course number one and two are related by a batch size factor right? The number of steps times my batch size is equal to the number of examples.

Now through some calculations open AI folks and others argue that this is the rough functional form that we should follow that the number of steps that you need and the number of examples that you need are inversely related with you know this Smin which is the minimum number of possible steps and the minimum number of examples sort of normalizing these two quantities. And so there's kind of a trade-off between these two right? If you want to minimize the number of examples you're going to have a lot of steps and if you want to minimize

the number of steps you're going to need a lot of examples right? That's a very natural trade-off right? And if you increase or decrease your batch size you're going to have trade-offs that go one side or the other.

Now they argue the goal now should be to balance these two sides. I want to make sure that my number of steps term and the number of examples terms are roughly balanced with each other and if you solve for that you're going to get this this particular solution which is that the critical batch size is the minimum number of examples that I need divided by the minimum number of steps that I need right? And these two are kind of estimated by fitting this to sort of the the observed data that I have.

This is going to balance both sides of the equation. It's going to give me a little bit more number of steps than optimal. It's going to require a little bit more examples than optimal but it balances the two sides and you can kind of think of this as improve optimizing the convergence rate without blowing up the number of steps that you you need. There's another added point that I won't really belabor because this gets even more into the weeds. You should really read the critical batch size paper if you're interested into this where one of their arguments is estimating this is quite complicated. You can do this by just looking at the ratio of the gradient covariance and the squared norm of your gradient. Um Okay. So why is that even in the scaling lecture right? Like this is this is a thing that is is not really related to scaling so far. The reason why this is in the scaling lecture is because the first of all the critical batch size is a reasonable batch size to pick. That's a batch size that balances sort of convergence rate and the size of the batch right? So if I want to pick my batch size equal to my critical batch size you might ask how does this thing scale as a function of my target loss right? Target loss can be thought of as compute right? If I want to have a model that is better and better and better right? What is the batch size that I I should use right? And the reason why this is in the scaling law lecture is as you go down in scale in the loss right?

So your loss is improving the amount of critical batch size increases and it increases in this very predictable way which is once again a power law right? So this is the the relationship that they argue which is roughly like inverse polynomial in batch size. So given this this is actually quite nice because what this is saying is if you have a really large scale training run where you go all the way to the right of this plot your batch sizes can be quite large right? And that kind of makes sense. As you get the closer and closer and closer to the minimum the value of variance reduction increases right? Because you're you're trying to optimize finer and finer and finer grained objects the noise matters a lot more once you're minimizing really really tiny differences.

Okay. So that's the batch size story. The other side right? So the two quantities remember that you're going to always have to tune is batch size and learning rate. The learning rate side of the story is also a little bit complicated. I'll go into this much more detail in the advanced scaling lecture. But learning rates generally shift. I think this is a good mental picture for how to think about learning rates in the standard world that that you start with which is let's say that you have just like a MLP right? Like just a standard neural network of some kind.

I'm not going to do depth scaling. I'm only going to do width scaling. Now if I'm doing width scaling the bigger my model right? The bigger my model the smaller my learning rate should be and the reason why my learning

rate should be smaller is because I have you know bigger more parameters and changing more things at once maybe I should move less right? Kind of makes sense. And so you and in fact there's a kind of rule of thumb that maybe you want to scale by one over the width. That's a very commonly known rule of thumb that you know you want to just decrease the learning rate regularly as a function of your width and that will give you a rule of thumb to scale your learning rate.

There is kind of another school of thought or another set of tricks that people do, which is that people also rescale the network in various ways. They will change the initialization sizes, and they will change kind of the step sizes of the optimizer or for various parts of the network in order to to force the model to have the same uh learning rate minimum across scales. Um and this is a idea um that is called P and others like it. Um some people have reported great success with these kinds of approaches. Um others have reported less success. The two ways basically of setting a learning rate um for large-scale runs really is kind of captured in this thought. In one way, you can try to estimate what these minima are and try to predict what your minimum will be. And the way in which the minimum changes is pretty predictable, so this is not a crazy idea. Or you can reparameterize your model to try to keep the learning rate minimum the same, and then just pick your best learning rate and go, right?

Those are the two philosophies to picking your learning rate. Both of them have been uh applied successfully at scale. There are large-scale training runs that have used both approaches. Um I think anecdotally um it does seem like more people are maybe favoring the scaling law approach, um but both are certainly viable approaches. Um I'll talk in detail about uh both of these next next next lecture. Okay. Um so, before I get into to kind of the last part um of this lecture where I want to talk about not quite pitfalls, but various you know details of of scaling law things.

Um one thing I want to end this section with is the idea that upstream and downstream performance can be quite different. So, if you look at something like, you know, uh log likelihood. This is from the ETE, you know, architectures paper. Um you know, you you find that perplexity and uh negative perplexity in this case, uh parameters are very correlated. Like the more parameters you have, the better your model very predictably, right? This is this is a very beautiful linear trend. Now, you're like, "Okay, this is great. Now, we're going to ship our best model NL-12." Um turns out NL-12 is not your best model. It was actually NL-32 XL, which was a much worse model in perplexity, right? This is a very common thing that does often correlated. Um this is probably one of the worst correlations that I've seen from upstream to downstream, but I think it's very very important to remember that scaling laws, generally speaking, are objects that you want to apply on the perplexity side. They're very clean, they're very regular, they're very predictable. But transfer from perplexity to downstream is a lot less certain than it might initially seem, right? This is a place where you want to be very cautious. Um And you know, as an anecdote, um I have I have former students that have gone in and done post-training at various places, and they always complain. They're like, "Oh, those pre-training people, you know, they hand you this model. They're like, 'The perplexity is good. It's all your problem now.'" But it's often, you know, the the problems have started at the at the pre-training side, right? So, you don't you don't want to be one of those people. You want to really think about transfer as well, not just uh focus on perplexities.

Okay. So, maybe this is surprising, maybe this is not, but the point that I want to drill in here is that, you know, before you do your big training run, you should have a rough idea of what you're going to see, right? Not just like, is it going to like train at all? In fact, if you do scaling laws, you should be able to actually predict, you know, fairly precisely the numerical values almost of how good your run is going to be, right?

If I choose one optimizer versus the other, what will the gain be? Like you should be able to predict that, right? Um so, the scaling law-based procedure is to say, "I'm going to train a bunch of smaller models. I'm going to establish some sort of a scaling law, you know, by fitting these lines onto log-log plots, and then if the fit is good enough, I'm going to have some confidence that the gap will persist, and then I'm just going to deploy that onto my large-scale training run, right?

This is the naive version of a scaling law proc- procedure, but I think one that is like reasonably accurate for how you might think about these things in practice, right? What else are you going to do? Um just sort of deploy a run out of nowhere? Probably not. Um Okay. I can pause here and and take some questions before I get into the the last part of uh of this lecture. Yes. For each of the data points on the scaling law, like how do you think about variance? Like how many runs do you do?

And is this aggregation like average or do you take minimum or maximum like that? Yeah, I think in most cases, if not all cases that I've seen, um those are singletons. And the reason why it's a singleton is because when you're looking at perplexities, perplexities are very very clean, right? Like, you know, these objects, if you rerun them, um you know, they're probably like the variance is in the second decimal place or something. Um because the the training data is very homogeneous, you've got a lot of it, your eval sets are very big. You really push the variance very very low on these things. Um but if you're doing something like, you know, um learning rate or critical batch size scaling laws, you will see some truly horrendous stuff. Um I do think people do some variance reduction, but it's actually not that common. As common as it should be, maybe.

Yes. What's the most people use like just downstream metrics um directly instead of using perplexity? Yeah, people do do scaling laws directly on downstream metrics, but that has the same problem as this. So, I think the the way to think about scaling laws um is to really reliably establish at least one thing. So, so the way to think about it is let's say I see something like this, right? I know or, you know, I have strong belief that this transformer line is going to continue. This is a very regular trend, right? And it might curve a little bit, but we do think there's probably some regularity here, right?

But instead, imagine this was a very noisy jaggedy line of this form. You would not really know what is going on. Like you would say, "I maybe believe that it will go on, but maybe it's a line, maybe it's a curve, right?" So, I think the general, you know, if I'm trying to to explain the philosophy of this stuff, is to say, "Okay, let's start with a low variance measurement, and then establish scaling regularity there, and then rely on, you know, some sort of belief about transfer or establish transfer, you know, elsewhere to try to show that that would transfer over to downstream."

Oh, yeah. So, some of these are for the test loss and some are for the train loss. When do you do either? I see.

Yes. Um you fit the test loss when, you know, you're you're doing good scientific practice like Kaplan 2021, but realistically, um all the uh the models except for like a very few, like the the infinite uh compute one and like the repetition stuff, unless you're doing some of that, um you're all you're doing one-pass SGD.

So, the generalization gap for all of these models is very very small, right? So, train and val are very similar. Um I've seen some pre-training codebases that don't even have validation loss, because if you're doing one pass, most likely your train and val are very very close. Um that's why a lot of these plots sort of exchange those two because they're all in the one-pass regime. Good question, though. Okay. All right. We will continue to move on to Sorry, I moved on too far backwards here. Okay.

So, now I'm going to talk about one of the most well-known uses of a scaling law, um even though you might not personally be, you know, uh using this, you've certainly used the rule of thumb at some point. Um and also, I think because this set of things will tell us something really interesting about the actual execution of a scaling law. So, okay. The motivating question is, do I want more data or bigger models, right? So, you know, the resource that you have, the resource that you have to spend is compute, right? Someone's going to hand you, metaphorically, certain amount of flops, right? And you're going to take those flops and you're going to spend them somehow, you know, on if you remember Percy's very first lecture, right? It's going to be data times the parameters is roughly your flops, right?

Linear in that. So, do I want more data or more models? Well, we know that if we take, you know, a teeny tiny model and we dump tons of data into it, that will just be wasted. That's kind of the uh you know, the dark purple line at the very top here. I've taken a teeny tiny language model and I've, you know, passed a huge amount of data into it, and this thing has been flat for a very long time. You know, this is a complete waste of compute, right? You would have much rather trained this big yellow model to the same amount of tokens. Or even earlier on, you know, if you're having the same amount of compute.

Now, you know, if you could sort of understand the interaction of how data amount and model size come together to sort of uh give you performance, if you understood that relationship, you could optimize this very precisely, right? Um and so, there are scaling laws that kind of are more advanced and relate kind of joint behaviors of objects. So, um almost simultaneously, um Kaplan and Rosenfeld both sort of proposed these kind of functional forms, which are roughly equivalent, um describing how the size of the model and the amount of data relate to the error.

Um Rosenfeld's is very simple. It's really just the sum of two inverse terms. These are two scaling laws that are added together. Um and Kaplan's is a little bit more complicated, but it's roughly the same idea, right? Um and if you look at think about the limits, this is very intuitive, right? If I have an infinite amount of data, right? Then what happens? Then this goes to zero. You're basically model size limited. You become a pure model size scaling law. If I send my model size to infinity, then I'm, you know, I'm going to be bounded by data.

I'm going to have a pure data scaling law, right? So, it makes sense if you take the limits of both of these. It's a often a a good idea if someone tells you a scaling law to take the limits of all the variables to understand uh the

behavior of the system. Now, the thing that's kind of cool is um Rosenfeld and many others that have proposed these kinds of joint scaling laws um show that that these things fit quite well even if you extrapolate. So, what you're going to do is you're going to fit on these green points, right? Which is a low compute regime and low data regime. You never see things that have either high data or high model sizes, right? And then you can extrapolate on the basis of these tiny neurons in this corner over here to essentially this high compute or high data high model size regime and you can very accurately predict the error based on your predictions.

Great sort of predictions, right? And so once you have that, now what you can do is to say, well, I have a fixed amount of flops subject to my flops constraints, minimize this, right? That's a simple non-linear optimization problem. You can do that to solve for this. Now, both Kaplan and Rosenfeld provide equations of this form. What sort of do they say? Well, if you look at Kaplan and they solve for this equation, they say something like this. Well, the amount of They always flip this. It confuses me.

The amount of data it should be C to the 0.73 and the amount of model size should be C to the 0.27. All right, is that reversed? I think it's reversed. Sorry. N N should be the amount of parameters and D should be the amount of data. And so tokens per parameter as decreases with C, right? So, as you increase the compute amounts, this prescription by Kaplan tells you to train bigger and bigger and bigger models.

And so if you were around in the days of GPT-3, it kind of shocks me to think that there are people who were not. You know, you would know that there was a period where everyone was training these like gigantic models, right? Like they were like hundreds of billions of parameters of models, like trillion parameter dense models, what have you? And part of that was driven by this, right? They saw GPT-3, they saw these like Kaplan predictions and they said, wow, I'm going to do my big training runs on these gigantic, you know, model size models. Um But, you know, a few years later in 2022, some folks at DeepMind, Hoffman and all came in and they said, well, actually these predictions are all terribly off, right? You've been training these giant models, these three stars over here, but actually those are way too big. What we should be training instead is a very different trend, this this line on the blue over here, and we should be training this teal star, which is a much better model, right? So, they they argued that we should be training much smaller models relative to this.

And I think, you know, of course, many of you or all of you probably know the Chinchilla paper, you know, the what the token multipliers are, right? It's 20 tokens to every parameter. But I think this is useful to go in in detail because the Chinchilla-Kaplan sort of disagreement and like all of this tells us something important about how we should be thinking about scaling laws, right? It's not just something where you like turn the crank and you get the answer. It's something that's actually careful and sensitive and we have to, you know, respect the process. Um Okay, so I'm going to walk through the Chinchilla paper and after I walk through the Chinchilla paper, I'm going to talk about why the Kaplan and Chinchilla papers disagree and what that means for you. Okay. So, the Chinchilla authors suggest three different ways of fitting scaling laws. I I really like this because it's a way of, you know, robustifying yourself to sort of modeling assumptions you may have made.

Um and each of these actually is a quite a different approach to estimating the trade-off between model size and data. Um and minus method three, which does differ a little bit, um they mostly agree. Like all one and two both

agree that the coefficients for data and for model size are roughly the same. 0.5 0.5 0.49 0.51. Method three does differ, you know, 0.46 and 0.54, but we'll talk about why later. Kaplan and all, of course, have a very big difference between the two, right? Which is why they were training big big models at big big compute sizes.

Method number one from Chinchilla is what I might call like the lower envelope method. And this is the method that, you know, was also being used partially by Kaplan to try to figure out what the right scaling was. And so what you do is you take all of your training runs. So, each one of these colored lines is a training curve, right? And what you want to do is you want to actually take the lower envelope of these training curves, right? Because a lower envelope point on this training curve, that means that for a particular flop, this is the best loss that I ever achieved, right? For that flop on any training run, right? So, I can take all of the lower envelope points at the very bottom here at the high compute regime and then I can plot all those points to see how big were the models that each of these flop points intersected, right?

Because each of these points is a training run with a model size. And if I scatter plot that, I will get a nice linear trend and that will give me one way of estimating the trip optimal trade-off between flops to parameters. And that gives one answer, which is for their compute budget, they should train a 67 billion parameter model, right? This is a nice and simple way, but very tricky to finding things like what is the lower envelope, so on and so forth. Like there there's definitely some tricky parts. Um Method two is isoflops. This is now very popular. It's also very easy and very robust. This is probably my personal favorite method.

And what you do is for this, you pick a bunch of different flops budgets, right? So, I have a different range of flops budgets, you know, going in factors of three in this case or even one three six. Um and what you do is for each flop budget, you're going to essentially sweep the space, right? So, here I have one variable, which is the parameter to data trade-off. So, for each flop budget, I'm going to sweep over the parameter to data trade-off, fixing my flops, right? So, I I double my data size, I half my model size, so on and so forth, right? And that will generate a sequence of training runs whose terminal loss is going to sweep out some sort of curve, right? So, each one of these colored sets of points is a fixed flop run. Each point is sweeping across different parameters and of course, it's implicitly sweeping over data as well, right? Now, the nice thing now is I can take the minimum over each of these runs or even fit a quadratic and take the minimum over the quadratics and then I can draw a line through all of the bottoms of the quadratics or take, you know, the bottom of the quadratics and plot them as a function of flops to to parameters and that will give me another prediction, which agrees very closely.

This is 63 billion parameters for my compute budget. Now, the final approach that we can do is actually the maybe most natural one as suggested by the Rosenfeld paper as well as the Kaplan paper because both of them sort of propose a hypothesized functional form for how model size and data relate to loss, right? So, if you have such a joint functional form, what you can do is you can take, you know, a whole bunch of training runs that you've done and you can fit, you know, your your loss landscape, right? Which is your hypothesized functional form and you can just do curve fitting to estimate what the coefficients are in your scaling law. This is the most brute force way. You have a hypothesized relationship between loss, data size and model size, train a bunch of models, do curve fitting, right? Hopefully fairly straightforward, you know, how how you do that. Of course, if

you're doing this method, how you do the curve fitting is very important, right? Now you're fitting these surfaces, there's many variables, it's kind of tricky.

You know, this thing also has some trickiness just like method one, right? So, it's not it's not a straightforward way the same way that maybe isoflops is. Okay. So now, you know, that we've gone through what Chinchilla does, it doesn't seem materially different than what Kaplan was doing, right? It's not like Kaplan was doing one crazy thing and then the Chinchilla authors did something like way smarter and you said, wow, why didn't we do that from the the start, right? They both did pretty reasonable stuff. But if we look at their scaling predictions, they are very different, right? And in hindsight, we generally, you know, tend to agree with the fact that the Chinchilla authors had more of the correct scaling, right? If you do your own Chinchilla style analysis, you will find probably scaling is much closer to Chinchilla.

Now, so why is there such a big difference when they're both fitting these like joint scaling laws? Okay, so this is I think where we get into kind of the messy realities of how scaling laws are made, right? Scaling laws can really change depending on the precise ways in which you're implementing the scaling laws, the hyperparameter settings that you're using and even like what the x-axis is. So, there's a a really nice paper. Oh no, I covered the authors. Okay, you can I I can paste this in the in the uh uh lecture Slack later. Um called Resolving Discrepancies in Compute Optimal Scaling of Language Models. Y Yi is the last author. He was a former student here. Um and they, you know, show basically first, they replicate the Kaplan result, you know, using roughly the Kaplan settings. And then they say, okay, well, you know, let's change how we count parameters.

They that shifts the curve a little bit. And then they say, okay, well, let's change how learning rate warm-up is done. That shifts the curve a little bit. And then let's tune the optimizer a little bit more. And then they get exactly kind of the Chinchilla result, right? So, their argument here is that there's a sequence of what seems like very minor decisions that in the end gives you this big gap between the Kaplan scaling law and the Chinchilla scaling law, right?

Um and I'll explain what each of these steps is, not that that's like the most important single thing, but also to emphasize to you how kind of important the details are here, right? So, the very first arrow where we go from Kaplan to the blue line, that is how we count parameters. So, remember in Kaplan I said, well, they saw this weird stuff if they included embedding parameters, right? So, they excluded embedding parameters. That's generally an okay thing to do, but one thing they also did that really messed them up was they also excluded the the last layer parameter counts, right? The the softmax ones because in many models, kind of the embedding and the last layer linear are dual to each other, right? Like they have the same shapes, right? The inside has vocab size by hidden dim, the output matrix has hidden dim by vocab size. And because they're the same shape, they said, "Well, let's just exclude both of those."

Turns out, whether you include or exclude those has a big material impact on the shape of the scaling law, right? They also found This is much more of kind of a oversight more than a genuine sort of difference in opinion that a lot of the models that were being trained in Kaplan were very small. In fact, they were so small that they were not actually converging by the time that the learning rate warm-up was done, right?

So, their learning rates were set very suboptimally. And because of that, you know, their models weren't fully converged. And then finally, they found that, you know, Kaplan et al. was fixing one big batch size, and those big batch sizes were suboptimal for the smaller models. And therefore, you know, if you tune that correctly and use varying batch sizes, you end up getting this sort of thing that agrees exactly with Chinchilla, right? So, I think each of these might seem like a very minor difference, but if you sort of change up these really minor you can get big shifts in the scaling law. And one thing I kind of like to emphasize is that scaling laws are kind of lower bounds in some sense, right? They're kind of saying, "If I continue this recipe and I scale it up, then this is what I will get, right?"

But if you're scaling up sort of a recipe that you don't want to scale up, like your warm-up is kind of crazy or your batch sizes are crazy, you're going to get bad scaling laws, right? So, you want to be as close to the proper sort of full run as possible. There's another paper that does another very interesting thing to try to study the difference between Kaplan and Chinchilla scaling laws. It's by Pearson and Song. And what they do, and I think this is very clever, is they don't train any models. They just take the Chinchilla training curves or like or the implied functional form of the Chinchilla training curves. They just plot them out, and they say, "Well, based on these, you know, implied training curves, which are simulated purely from Chinchilla, they generate what the implied training curves will be for Kaplan if you ignore the embedding parameters and scale way down, and what happens if you use the total parameter count instead." And what they find is really two effects. One is that Kaplan is operating at a much lower compute scale, so they're very sensitive to small changes. And actually, ignoring the non-embedding parameter is a sufficiently big change to mess up the result relative to Chinchilla. So, their their argument is slightly different from Yair and others, which is that here actually, it's the size of the compute scale plus the slight nonlinearity induced by non-embedding parameters that's causing this this disagreement, right?

So, this is really, you know, starting to get into the weeds, but I hope you kind of appreciate, you know, the importance of paying attention to detail, right? And I think a lot of the the work that goes on in establishing reliable scaling laws is about figuring out what the right regime is so that you can get reliable scaling across, you know, compute scales. The final thing, and this is really just more of something that I just find amusing more than anything else, which is that, you know, Chinchilla is this paper that kind of takes down the Kaplan paper in a way, right? Like, "Oh, you guys were very wrong."

But there was this one remaining mystery, which is that Chinchilla method three never agreed with Chinchilla method one and two, right? And the numbers look small, and the authors are are very unperturbed about this, but I think the implications are quite nontrivial. Like, if you look at this, method one and two basically say, you know, the amount of parameters and tokens should scale together, right? So, there should be a fixed ratio between the two. This is where kind of the Chinchilla factor of 20 comes from.

If you look at, you know, number three, the exponents are different. This means if you scale up your compute, eventually, actually in this case, you're going to have way more tokens than you are going to have parameters, right? So, it's going to be actually a very different scaling conclusion asymptotically than the other ones. And so, turns out that you know, I was certainly not the only ones perturbed about this, except some folks at Epoch AI

did something about being perturbed.

They couldn't get either the raw data or the code. So, what they did was they extracted the data from plots in the paper, and they refitted the method three regressions. Like, they extracted the data from the paper, and given that data, they fitted that sort of 2D surface or sorry, 3D surface that was proposed. And then, what they showed is actually the original paper must have underfitted the original data. And if they redid sort of the curve fitting, they could get much lower losses relative to what sort of the the proposed scaling law in the Chinchilla paper. And in fact, if you follow the refitted model, you got almost exactly the you know, D over 20 is 20 rule of thumb from the Chinchilla paper, right? Which is really a funny conclusion, I think, to the Chinchilla paper, where in the end, it turns out that, you know, the authors were more right than they knew, right? Actually, they they had made a mistake, and that was the only reason why method three disagreed from methods one and two. So, it's kind of a nice sort of conclusion to know that the method was robust after all.

Okay. So, that's the the end of the Chinchilla saga in some ways, but I will end this section by saying you probably don't want the Chinchilla factor. At this point, I think this is a pretty common, you know, knowledge thing. But if you're training a production model, right? You don't really care about saving training compute for the most part. You can look at all sorts of external surveys that people have done of how frontier labs are spending their compute. Most of the compute is not going into training runs, right? Most of the compute is actually going into R&D and serving, right? And given that, especially for serving, what you want to do is you want small models that are capable, right? You do not want big, bloated models that cost a lot to serve, even if that minimizes training cost.

And so, what you really want is something that you would call as overtrained. I put that in quotes because really, overtraining is what we want. That's the right amount of training. And if you look at all the models that were released, you know, in the early days, like GPT-3, this was undertrained for sure, right? Three tokens per parameter. Chinchilla gets to the modern ratio of 20 per parameter. And then for a while, people were at 20 per parameter. And this is the era, I think, where there wasn't that much serving of these models, right? The models were cool, but it wasn't being served at scale. And then as you sort of start to move to eras where, you know, serving becomes very real, now you're overtraining these models, and now you've moved into MoEs, and you're making all these tradeoffs to optimize inference serving, right?

So, I think Chinchilla is important not because I think the 20:1 ratio is, you know, the one golden ratio. If you're doing stuff for research, it might be. But I think the reason why it's important is because it tells us quite a bit about how we should be fitting scaling laws and how to think about scaling laws. Okay. One final thing I'll end with Chinchilla-wise and and otherwise for scaling is isoflops have been one of the things that I think as a research tool have endured.

Isoflops are very easy to execute, right? You fix a flop budget, and then you sweep over all of the other degrees of freedom to see kind of what the surface looks like. And this gives you in general fairly reliable readings of how different parameters vary as a function of your of your of your free parameters. You know, people have done this for or actually actually my one of my students did this for diffusion models. You know, the MoE study

I talked about was a isoflop style design. If you're ever in a situation where you're thinking, "How am I going to sort of decide all these tradeoffs?"

isoflops is always a good default. Okay. So, to to wrap up everything for today, you know, there's this idea that, you know, we have this log-linear regularity between the amount of resources we put in and the performance we get out, right? It extends to a wide variety of things, model parameters, compute, sparsity level of your MoE. And that lets you do a lot of the things that I was talking about earlier as like arbitrary choices, but do it in a much more evidence-driven way, hopefully.

And so, this allows us to have engineering at scale without explicitly relying on doing big model training runs, okay? And hopefully, you remember, I think, all the different components here. Data scaling is this very natural object. Just the exponent is a little bit strange. Model scaling allows you to do all these cool engineering things. And then finally, scaling is this very cool, robust predictor, let's call it, for how to try to do engineering at large scale.

And thanks. Next lecture, I think, will be Percy on inference, but after that, we will return to more advanced niche scaling topics.